

ATARI ST+STE

# BIEN DEBUTER EN GFA BASIC

VERSIONS 2.0 ET 3.0 INCLUSES  
LES NOUVEAUTES DE  
LA VERSION 3.5

EDITIONS MICRO APPLICATION



LIVRE DATA BECKER

# **BIEN DEBUTER EN GFA BASIC**

**V E R S I O N S**

**2 . 0 E T 3 . 0**

**I N C L U S E S**

**MICRO APPLICATION**

58, rue du Fg Poissonnière  
75010 PARIS

© Reproduction interdite sans l'autorisation de  
**MICRO APPLICATION**

'Toute représentation ou reproduction, intégrale ou partielle, faite sans le consentement de **MICRO APPLICATION** est illicite (Loi du 11 Mars 1957, article 40, 1er alinéa).

Cette représentation ou reproduction illicite, par quelque procédé que ce soit, constituerait une contrefaçon sanctionnée par les articles 425 et suivants du Code Pénal.

La Loi du 11 Mars 1957 n'autorise, aux termes des alinéas 2 et 3 de l'article 41, que les copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à l'utilisation collective d'une part, et d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration'.

ISBN : 2-86899-169-6

© 1988 **DATA BECKER**  
Merowingerstraße, 30  
4000 Düsseldorf - RFA

*Auteur : Hans-Georg Schumann*

*Traduction Française assurée par Monsieur et Madame Baudin*

© 1989 **MICRO APPLICATION**  
58, rue du Fg Poissonnière  
75010 PARIS

Collection dirigée par Mr Philippe OLIVIER  
Edition réalisée par Frédérique BEAUDONNET

Atari ST et ST sont des marques déposées de Atari Corp.  
GEM est une marque déposée de Digital Research Inc.  
GFA Basic est une marque déposée de GFA Systemtechnik

# Sommaire

## **Partie 1 : Introduction au GFA Basic .....1**

---

<b>1. Pour s'échauffer .....</b>	<b>3</b>
1.1. Sauvegarde et mise en route .....	3
1.2. Editeur et menus .....	8
<b>2. Votre premier programme .....</b>	<b>13</b>
2.1. Question et réponse .....	13
2.2. Ça va bien, ça va mal ... ..	17
2.3. Une note plus personnelle ? .....	19
2.4. En résumé .....	23
<b>3. Une autre perception du Basic .....</b>	<b>27</b>
3.1. Pourquoi justement un GFA Basic ? .....	27
3.2. Interpréteur ou compilateur ? .....	28
3.3. To GO or not to GO... ..	30
3.4. Pour votre confort .....	32
3.5. Résumé .....	34

## **Partie 2 : Le savoir de base en GFA Basic .....35**

---

<b>4. Les sauts et les boucles .....</b>	<b>37</b>
4.1. Les sauts sont possibles... ..	38
4.2. ...mais ils ne sont pas indispensables .....	40
4.3. Aussi longtemps...jusqu'à ce que... ..	44
4.4. Résumé .....	50
<b>5. Les répétitions .....</b>	<b>53</b>
5.1. Pour savoir compter, il faut apprendre .....	53
5.2. Monter et descendre .....	57
5.3. Des boucles sans fin ? .....	60
5.4. Résumé .....	63



<b>6. De la manipulation des données</b>	<b>67</b>
6.1. Variables et constantes	67
6.2. Les types de données	70
6.3. Définition et débuts des valeurs	74
6.4. Les opérateurs	77
6.5 Résumé	78
<b>7. Les procédures</b>	<b>81</b>
7.1. Les paragraphes	81
7.2. Nouvelles instructions	84
7.3. Local ou global ?	88
7.4. Réutilisation des procédures	91
7.5. Résumé	92
<b>8. Les fonctions</b>	<b>95</b>
8.1. Pourquoi des fonctions	95
8.2. Un peu d'acrobatie sur les touches	98
8.3. Arithmétique des signes	101
8.4. Résumé	106
<b>9. Graphisme et son</b>	<b>109</b>
9.1. Les points et les lignes	109
9.2. Rond ou carré ?	112
9.3. Oeuvres complètes	115
9.4. L'image ou le son ?	119
9.5. Résumé	121

## **Partie 3 : Le GFA Basic pour les utilisateurs ....123**

---

<b>10. Les structures de programme</b>	<b>125</b>
10.1. Un menu	125
10.2. Options et choix	128
10.3. De cas en cas	130
10.4. Répétition de la possibilité de choix	134
10.5. Résumé	138
<b>11. Les menus, les colonnes et les fenêtres :</b>	
<b>une bonne optique</b>	<b>139</b>
11.1. Tout doit tenir dans un cadre	139
11.2. Encore un menu	141
11.3. Listing et folding	144
11.4. Vous aimez les fenêtres ?	148
11.5. Résumé	150

<b>12. Collecte des données .....</b>	<b>151</b>
12.1. Une banque sans données .....	151
12.2. Abondance de données ? .....	154
12.3. Enregistrer ! .....	157
12.4. Résumé .....	160
<b>13. Manipulation des données après formatage .....</b>	<b>161</b>
13.1. Il nous faut un formulaire .....	161
13.2. La forme et le contenu .....	163
13.3. L'entrée des données... ..	165
13.4. ... et la sortie des données .....	170
13.5. Chirurgie plastique .....	172
13.6. Résumé .....	175
<b>14. Utilisation des disquettes .....</b>	<b>177</b>
14.1. Ouvrir et fermer .....	177
14.2. Un coffre-fort pour la banque .....	180
14.3. Sauvegarder et recharger .....	183
14.4. Transfert de données .....	186
14.5. Résumé .....	189
<b>15. Traitement des données .....</b>	<b>191</b>
15.1. Il faut pouvoir apporter des corrections... ..	191
15.2. .... et aussi des améliorations .....	194
15.3. Classement des données .....	198
15.4. Toutes sortes de choses bien utiles .....	200
15.5. Vous allez savoir marcher tout seul ? .....	203
15.6. Résumé .....	205
<b>Partie 4 : Au secours ! .....</b>	<b>207</b>

---

<b>16. Vue d'ensemble sur le GFA Basic .....</b>	<b>209</b>
16.1. Le vocabulaire .....	210
16.2. Du GFA 2.0 au GFA 3.0 .....	226
<b>17. Une mine de renseignements bien utiles .....</b>	<b>227</b>
17.1. Les mots clés du GFA Basic .....	227
17.2. Pour éviter les erreurs .....	230
17.3. Pour conclure .....	231

## **Partie 5 : Annexes .....233**

---

### **A : Les commandes éditeur ..... 235**

1. Déplacement du curseur .....235
2. Insérer et effacer .....235
3. Manipulation des blocs .....236
4. Autres opérations .....236

### **B : Les menus du GFA Basic ..... 237**

1. Pour appeler le programme : .....237
2. Le menu principal : .....237
3. La version 3.0 comprend en plus les possibilités suivantes : .....238

### **C : Tableau des caractères ASCII de l'Atari ..... 239**

### **D : Le GFA Basic V 3.5 ..... 243**

### **E : Index des mots-clés ..... 247**

## Introduction :

Le GFA Basic est devenu actuellement le langage de programmation standard sur le monde de l'Atari ST. Il permet à un utilisateur comme à un programmeur de tirer pleinement parti de l'Atari ST.

Ce Basic en est maintenant à sa troisième version, encore plus confortable et plus puissante, pouvant servir aussi bien pour une simple initiation à la programmation que pour des projets plus élaborés.

Qu'allez-vous trouver dans ce livre ? Dans la première partie vous ferez vos premiers pas en programmation, vous apprendrez à manipuler l'éditeur GFA et les commandes les plus importantes.

Dans la deuxième partie, nous traiterons des bases du GFA Basic : les boucles, la manipulation de données, les procédures et les fonctions ainsi que la création graphique et sonore.

Dans la troisième partie vous n'êtes plus un débutant, mais un utilisateur expérimenté. Vous faites connaissance avec de nouvelles commandes du riche vocabulaire GFA et apprenez à manipuler le clavier, l'écran, l'imprimante et les unités de disquettes. Vous aurez également un aperçu de la programmation GEM.

La quatrième partie est faite pour vous secourir lorsque vous aurez besoin d'aide, ce qui ne manquera pas d'arriver ; elle contient tout d'abord une vue d'ensemble de toutes les commandes tirées du vocabulaire GFA Basic traitées dans ce livre ainsi que quelques remarques sur les versions 2.0 et 3.0. Viennent ensuite quelques conseils pour éviter les fautes que vous pourriez commettre en tant que débutant ou utilisateur du GFA Basic. Un glossaire est prévu pour faciliter la recherche de certains mots-clés importants.

En annexe se trouve l'essentiel sur ce qui concerne l'éditeur du GFA Basic et ses menus. De plus, vous y trouvez un tableau des codes ASCII de l'Atari.

# **Partie 1**

## **Introduction au GFA Basic**





# Chapitre 1

## Pour s'échauffer

**D**ans cette section, nous allons rester au niveau le plus simple de l'apprentissage. Nous n'allons pas faire connaissance du GFA Basic de façon brutale. Après avoir manipulé un peu l'éditeur GFA, vous ferez votre premier programme. Nous nous entretiendrons ensuite du Basic en général et du GFA Basic en particulier.

On dit du langage Basic qu'il suffit de connaître quelques commandes pour pouvoir déjà se mettre directement à programmer. Sans avoir beaucoup à réfléchir - selon le proverbe "l'expérience concrète passe avant l'étude abstraite" - on peut rapidement confectionner de petits programmes. C'est pourquoi le Basic (abréviation de "*Beginners' All purpose Symbolic Instruction Code*") semble être le langage idéal de programmation pour les débutants.

### 1.1. Sauvegarde et mise en route

Allons-y ! C'est certainement avec ces mots que vous vous êtes assis devant votre ordinateur, avec ce livre ouvert à vos côtés. Mais sachez tout d'abord que dans ce domaine, on ne peut foncer tête baissée : vous devez apprendre à programmer de façon réfléchie. Vous ne constaterez pas tout de suite qu'il est plus rentable ici de faire passer "*l'étude abstraite avant l'expérience concrète*". En effet, une bonne analyse préalable ne se rentabilisera que lors de la confection de programmes longs et complexes. Et c'est justement là que le GFA Basic vous fournira une aide précieuse !

Branchez maintenant votre ordinateur comme vous le faites habituellement. Mettez la disquette système dans le lecteur. Nous allons tout d'abord nous entendre sur quelques termes que j'utiliserai souvent dans cet ouvrage. En

premier lieu ce que l'on appelle "*cliquer avec la souris*", qui signifie : appuyer sur la touche gauche de la souris. Essayez : amenez la flèche avec la souris sur une icône et cliquez une fois. Une icône est la représentation graphique de votre disquette sur l'écran. Nous verrons qu'il y a de la même façon une icône pour les programmes, une icône pour les dossiers etc...

Le symbole apparaît en inversion vidéo. C'est ce qu'on appelle le simple clic, permettant d'activer un symbole. Il existe de plus le double-clic qui sert par exemple à lancer un programme ou à afficher le contenu d'un fichier.

Cliquez deux fois rapidement sur l'icône de l'unité de disquettes : le contenu de la disquette système doit s'afficher sur l'écran. Si vous n'avez pas encore fait grand chose avec votre Atari, il peut arriver que le double clic ne fonctionne pas immédiatement. Ceci vient du fait que vous n'avez sans doute pas appuyé assez vite consécutivement, mais c'est en forgeant qu'on devient forgeron.

Avant de commencer, nous allons confectionner une disquette de sauvegarde de la disquette originale du GFA Basic. Cela nécessite une disquette vierge (si possible neuve), qu'il faut tout d'abord préparer, c'est à dire formater. Pour cela, placez la disquette dans l'unité A, activez l'icône par un simple clic (pas de double clic !) et cliquez sur *formatage...* dans le menu fichier de la barre de menus. Dans le milieu de l'écran apparaît alors un message d'alarme vous prévenant que toutes les données de la disquette A vont être effacées.

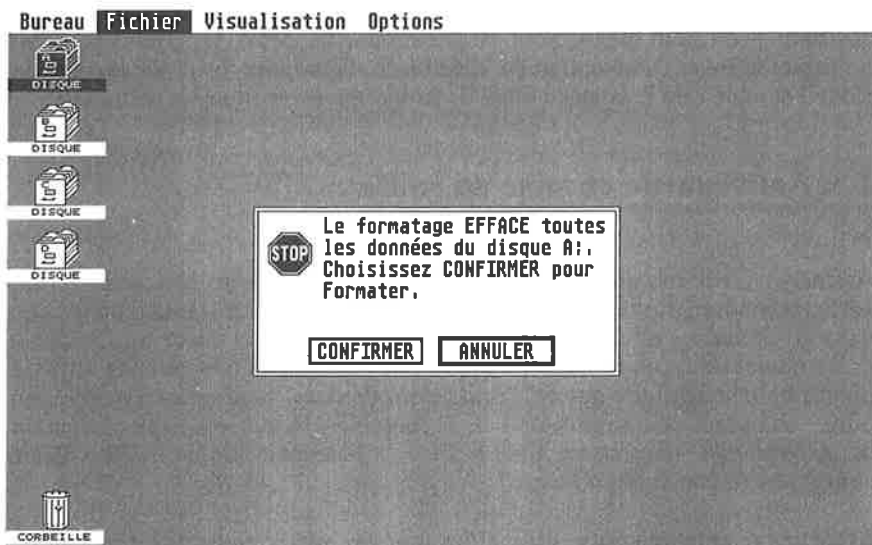


Figure 1 : Message d'alarme

Quand les boîtes d'alarmes apparaissent, un des boutons est entouré d'un cadre beaucoup plus noir.

Cela signifie que l'appui sur la touche <Return> suffit pour le sélectionner. Vous pouvez bien sûr l'activer en cliquant dessus avec la souris.

Après vous être assuré de nouveau qu'il y a vraiment une disquette vierge dans l'unité A, vous cliquez sur *Confirmer*, ce qui fait apparaître une fenêtre de dialogue.

Si la zone *simple face* apparaît en inversion vidéo, vous cliquez tout simplement sur *formater*.

Si c'est la zone *double face* qui apparaît en inversion, cliquez d'abord sur *simple face* puis sur *formater*.

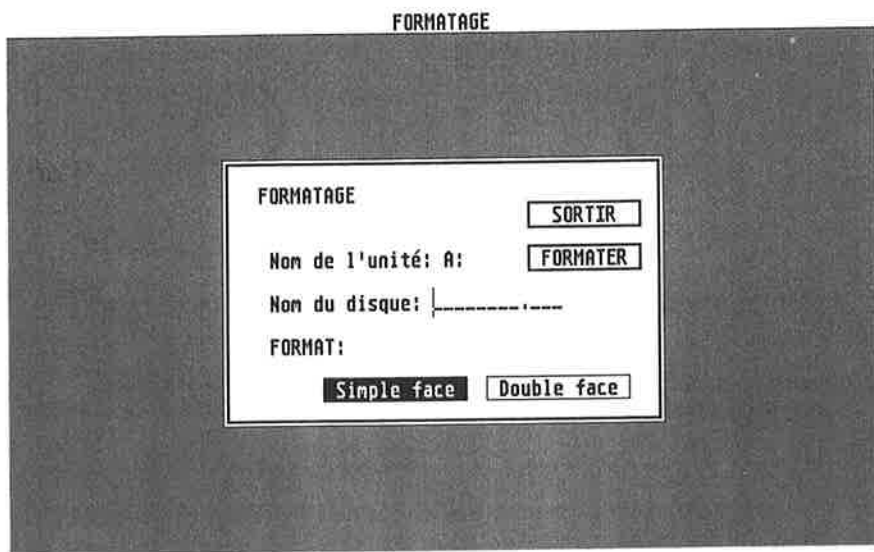


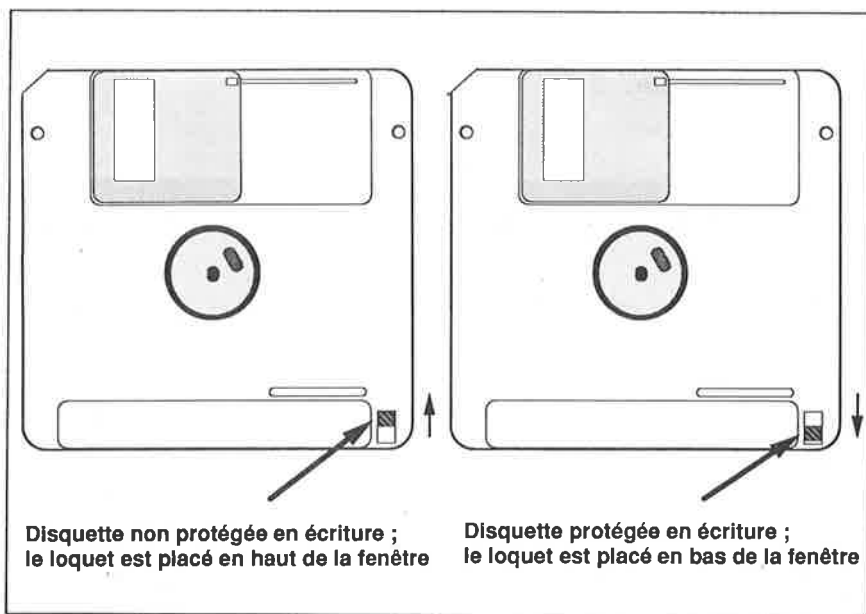
Figure 2 : Fenêtre de dialogue formatage

La disquette est en cours de formatage, l'état d'avancement du processus est figuré par un rectangle qui se remplit de noir. Après un certain temps, vous recevez le message vous annonçant le nombre d'octets disponibles. Si celui-ci n'est pas égal à 357376, répétez l'opération de formatage avec une autre disquette vierge.

Quittez la fenêtre de formatage en cliquant sur *Sortir*.

Nous allons maintenant procéder à la copie de la disquette originale de GFA Basic.

1. Pour ne pas effacer la disquette originale par erreur, il faut la protéger contre l'écriture. Pour cela, posez-la sur le dos, vous apercevez en haut à gauche un loquet couissant que vous placez de telle sorte qu'on puisse voir une petite fenêtre. La disquette est alors protégée contre toute écriture. Sur la disquette vierge formatée, cette fenêtre doit être fermée. La figure suivante vous montre la position du loquet :



2. Insérez maintenant la disquette originale de GFA Basic dans l'unité A. Si vous avez deux unités de disquette, introduisez de plus la disquette que vous venez de formater dans l'unité B.
3. A l'aide de la souris, amenez la flèche sur l'icône de l'unité A et cliquez : l'icône apparaît en inversion vidéo. Maintenez la touche gauche de la souris appuyée et faites glisser l'icône A sur l'icône B.
4. Un message d'alarme vous avertit alors que le processus de copie efface toutes les données contenues sur la disquette B : cliquez sur *Confirmer*

5. Dans la fenêtre suivante cliquez sur *Copier*

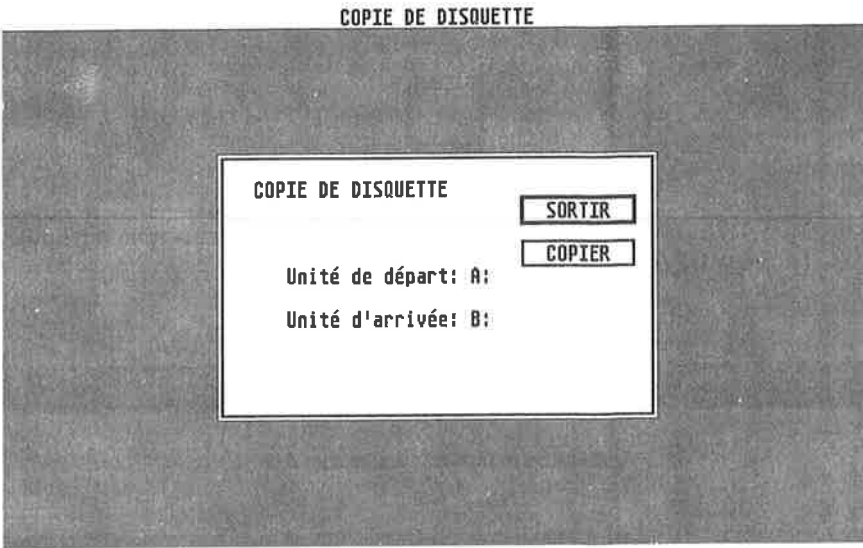


Figure 3 : Copie

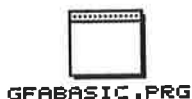
6. Si vous ne disposez que d'un lecteur de disquette, une fenêtre vous demande d'insérer la disquette sur laquelle vous voulez copier, donc celle que vous venez de formater. Durant le processus de copie, vous voyez un dessin qui vous montre l'état d'avancement du processus de copie. Lorsque vous ne disposez que d'un lecteur, vous devez intervertir plusieurs fois la disquette source et la disquette cible : suivez les instructions données par les diverses fenêtres qui s'affichent.
7. Lorsque le traitement est terminé, vous voyez réapparaître la première fenêtre de dialogue portant l'indication : *Copie de disquette*. Vous cliquez cette fois sur *Sortir* ce qui met fin au processus de copie.

Après avoir ainsi confectionné une copie de travail, rangez soigneusement la disquette originale, dont vous ne vous servirez plus que dans le cas où vous auriez détruit toutes vos copies de travail !

La disquette de travail devrait maintenant se trouver dans l'unité A.

Si vous n'avez pas encore ouvert la fenêtre permettant d'afficher le contenu de la disquette, rattrapez-vous en double-cliquant sur l'icône de l'unité A.

Parmi toutes les icônes qui s'affichent , vous devez alors pointer avec la flèche sur l'icône intitulée :



Un double-clic permet alors de charger le GFA Basic, et vous vous retrouvez dans l'éditeur GFA :



Barre de menus dans la version 3.0



Barre de menus dans la version 2.0.

Tout en haut s'affiche une barre des menus, encadrée dans la version 3.0 à gauche par le symbole-Atari et à droite par l'horloge en dessous de laquelle s'affiche le numéro de la ligne. Chacun de ces menus peut être ouvert en cliquant dessus avec la souris ou en appuyant sur les touches de fonction : les menus situés sur la ligne inférieure sont accessibles par les touches < F1 > à < F10 > , ceux de la ligne supérieure par < Shift > < F1 > à < Shift > < F10 > . Il en va de même pour la version 2.0.

## 1.2 Editeur et menus

L'éditeur GFA n'est pas vraiment un traitement de texte, il n'est pas conçu pour écrire vos lettres. L'éditeur teste chaque ligne d'écriture pour constater s'il s'agit bien d'un texte écrit en BASIC et sans erreur de syntaxe. Vous pouvez vous en convaincre vous-même en écrivant par exemple :

L'éditeur GFA Basic tient-il ce qu'il promet ?



Après quoi vous terminez en appuyant sur la touche < Return > . L'indication *faute de syntaxe* vous montre que votre phrase ne correspond pas aux règles du GFA Basic. Ce n'est donc pas la bonne façon de procéder avec l'éditeur GFA !

A l'aide de la touche curseur (flèche à gauche) positionnez-vous au début de la ligne, inscrivez une apostrophe (') et appuyez sur < Return > . Cette fois l'éditeur ne rechigne plus. Car ce petit signe lui indique que ce qui suit ne le concerne plus ! En faisant commencer toutes les lignes par cette apostrophe, vous pourriez saisir n'importe quel texte. Vous parviendriez donc à écrire votre lettre par ce moyen !

Essayez maintenant d'écrire quelques lignes afin de vous familiariser avec l'éditeur. Remarquez par la même occasion que le petit compteur de ligne en haut à droite (dans la version 3.0) compte les lignes que vous saisissez depuis le chiffre 0. Introduisez sans scrupule quelques fautes d'orthographe : l'éditeur ne remarque rien ! Avant de les corriger, jouez un peu avec le curseur.

Vous pouvez le déplacer à l'aide des touches fléchées à l'intérieur du texte, de haut en bas et de droite à gauche ; vous ne pourrez le déplacer sur toute la surface de l'écran qu'après l'avoir remplie de texte. La souris vous permet naturellement de placer le curseur n'importe où dans le texte, il vous suffit d'amener la flèche à l'endroit voulu et de cliquer.

Vous pouvez corriger les fautes de frappe à l'aide de la touche < Delete > ou < Backspace > : constatez-vous la différence de l'effet engendré ? Vous pouvez insérer directement ce que vous avez oublié ou ce que vous voulez rajouter car vous vous trouvez en mode écriture (Insert).

Voyons un peu ce qui arrive si vous cliquez dans la barre de menu sur Insert ou si vous actionnez la touche < F8 > : le mot Insert est remplacé par Overwr. Vous ne vous trouvez plus en mode "écriture" mais en mode "remplacement".

Comme vous voyez, vous ne pouvez plus insérer de texte, mais les lettres existantes sont remplacées directement par les nouvelles.

Ces instructions-éditeur pourraient à vrai dire vous suffire ; mais il serait dommage de ne pas faire appel aux autres possibilités de l'éditeur GFA :

- ✱ Vous pouvez par exemple atteindre directement le début ou la fin d'une ligne en combinant les touches < Control > < ← > ou < Control > < → >

- ❖ La touche <Insert> vous permet d'insérer une nouvelle ligne. En actionnant <Control> <Delete> vous pouvez effacer une ligne et (dans la version 3.0) la rappeler par <Control> - <U> si vous venez de gommer sans avoir rien fait d'autre !
- ❖ <Control> <Clr/Home> ou <Control> -Z vous permettent de placer directement le curseur en début ou fin de texte.

Ce ne sont là que quelques possibilités, et vous en apprendrez d'autres au cours de vos exercices de programmation. Vous trouverez d'ailleurs une liste complète des fonctions-éditeur dans l'annexe A.

Avant de vous lancer dans votre premier programme, vous devriez encore prendre connaissance des quelques informations suivantes concernant les possibilités offertes par la barre de menus pour la sauvegarde ou le chargement sur/depuis un fichier ou un disque dur d'un programme écrit en GFA Basic en utilisant :

Save	ou	<Shift> <F1>
Save,A	ou	<Shift> <F2>
Load	ou	<F1>
Merge	ou	<F2>.

Vous verrez apparaître à chaque fois une fenêtre de dialogue vous demandant de préciser le nom du fichier que vous appelez ou dans lequel vous voulez écrire.

*Save,A* vous permet de sauvegarder le texte de votre programme tel que vous l'avez saisi dans l'éditeur au signe près : LST est l'abréviation de Listing. Le listing du programme peut alors être relu par un logiciel de traitement de texte comme par exemple Textomat ST par exemple.

*Save* permet de sauvegarder votre programme écrit en GFA Basic de façon codée, c'est pourquoi dans la version 3.0 vous voyez apparaître l'extension GFA. La version 2.0 fait appel à un autre encodage : les programmes sont enregistrés comme des fichiers BAS. L'encodage permet de raccourcir le temps de chargement et la taille des programmes longs, mais le listing n'est plus lisible que par l'éditeur GFA.

*Load* a l'effet inverse : il vous permet de charger dans l'éditeur les fichiers GFA (ou BAS pour la version 2.0). *Merge* s'utilise pour les fichiers LST. Cette option vous permet d'ajouter des programmes dans l'éditeur sans écraser celui qui s'y trouve : mais ceci ne fonctionne qu'avec des fichiers LST !

Si vous chargez des programmes dont vous n'êtes pas l'auteur ou qui ont été élaborés avec un autre éditeur, il peut arriver qu'une ligne commence par une double-flèche allongée : le GFA Basic ne comprend pas cette ligne erronée.

*Run* ou <Shift> <F10> permettent de lancer le programme. *Test* ou <F10> servent à tester la bonne structure du programme. Lorsqu'un programme s'est entièrement déroulé et que vous revoyez son listing dans l'éditeur, *Flip* ou <F9> vous permettent de visualiser la situation laissée par le programme sur l'écran.

*Direct* ou <Shift> <F9> ou plus simplement <Esc> vous replacent dans le mode écriture directe : les lignes de menu disparaissent ainsi qu'un texte de programme éventuel et vous voyez apparaître la suite de signes

OK >

Tapez alors :

p 1+1

et terminez en actionnant la touche <Return> ou <Enter>. Vous pouvez naturellement demander quelque chose de plus compliqué ! La lettre "p" est l'abréviation de PRINT et signale en GFA Basic à votre Atari ST qu'il doit afficher quelque chose sur l'écran.

Si par hasard vous devez effectuer une petite addition et que vous n'avez pas sous la main votre calculatrice de poche, vous pouvez vous en sortir en vous plaçant sous *Direct*. Dans la version 3.0, pour revenir à l'éditeur GFA, pressez simultanément les touches : <Control> <Shift gauche> <Alternate> et dans certains cas (par exemple lorsque l'on utilise des menus déroulants (on menus), il faudrait appuyer en plus sur <Return>. Vous pouvez aussi taper END ou EDIT pour revenir aux menus GFA.

*Llist* ou <F3> vous permet de sortir sur papier le texte de votre programme par le biais d'une imprimante. *New* ou <F4> vous permet d'entrer un nouveau programme en effaçant l'ancien. Enfin vous pouvez quitter le GFA Basic par *Quit* ou <Shift> <F3>.

Dans la version 3.0 vous pouvez cliquer sur le symbole Atari en haut à gauche, ce qui fait apparaître un nouveau menu, vous donnant accès à des accessoires éventuels comme par exemple le réglage du panneau de contrôle ou de l'imprimante. En pointant sur *GFA Basic* vous ouvrez un menu :



Figure 4 : le menu figurant sous "GFA Basic"

*Save* et *Load* correspondent aux options du menu principal. Nous ne nous intéresserons plus maintenant qu'à *Deflist* qui vous permet de fixer le mode d'écriture de vos programmes. Vous avez deux possibilités :

- 0 L'éditeur écrit en lettres majuscules tous les mots du vocabulaire du GFA Basic et en minuscules les mots que vous définissez vous-même quel que soit votre mode d'écriture ; ce code est présélectionné par défaut lors du lancement du GFA Basic version 3.0
- 1 L'éditeur fait commencer tous les mots par une majuscule suivie de minuscules, qu'ils soient tirés du vocabulaire GFA Basic ou de votre création ; ce mode est présélectionné lors du lancement du GFA Basic version 2.0.

Notez que lorsque la valeur 0 est proposée, le choix en cours est 1 et inversement. Cliquez sur l'une des deux valeurs indiquées (0 ou 1). Vous retournez directement à l'éditeur.

En plus de ce *formatage* du texte, ajoutons que les mots saisis entre guillemets (") sont du texte et ceux entre apostrophes (') des commentaires. Les deux possibilités restantes (options 2 et 3 du menu) ne vous servent pas pour l'instant et vous n'avez donc pas encore à en prendre connaissance.

En cliquant sur *Editor* vous retournez dans l'éditeur, si vous n'y êtes pas déjà. Je crois que cela suffira pour un premier contact. Nous ferons appel aux autres options lorsque nous en aurons besoin ; vous trouverez de plus amples explications sur les menus dans l'annexe B.

## Chapitre 2

### Votre premier programme

Bonjour, comment vas-tu ?	
Bien ?	- je m'en réjouis !
Mal ?	- j'en suis bien peiné !

Votre Atari ST est-il aussi prévenant avec vous ? Non ? Alors apprenez lui les bonnes manières, et sans tarder ! Un ordinateur ne dialoguera avec vous que si vous lui faites comprendre clairement où vous voulez en venir ! Sans les instructions adéquates il ne fera rien, ou en tout cas pas ce que vous attendez de lui.

#### 2.1. Question et réponse

C'est à vous maintenant ! Donnez vos instructions ! Que doit faire votre ordinateur ? Il doit vous saluer par un *bonjour, comment vas-tu ?*. Vous voulez d'abord essayer par la manière douce ? le saluer de préférence par un gentil

bonjour
---------

dans l'espoir qu'il va vous répondre sur le même ton ? Mais lors de vos essais avec l'éditeur GFA vous avez déjà constaté qu'il n'accepte pas n'importe quel texte. Votre *bonjour* ne vous mènerait qu'à la réponse que vous connaissez déjà *erreur de syntaxe*, et même une apostrophe ne vous avance pas plus ici.

Vous devez user d'un autre ton. Votre ST attend visiblement de vous, une indication claire de ce qu'il doit faire. Concrètement cela veut dire que votre ordinateur doit écrire sur son écran.

Vous devez donc lui dire :

```
ECRIS "Bonjour, comment vas-tu ?"
```

surtout pas dans votre langue de tous les jours, mais dans celle qu'il comprend, le GFA Basic pour ce qui nous concerne. Effacez donc votre gentil *bonjour* grâce à <Control> <Delete> et ordonnez :

```
PRINT "Bonjour, comment vas-tu ?"
```

Enfin votre ordinateur peut vous dire :

```
"Bonjour, comment vas-tu ?"
```

Vous avez déjà fait la connaissance de PRINT dans le chapitre précédent. Il signifie *imprime* mais aussi *dis* ou *écrit* : c'est ce qui incite votre ordinateur à afficher quelque chose sur son écran qui est en quelque sorte sa façon de vous parler. Et ce quelque chose est précisément le texte placé entre les guillemets :

```
"Bonjour, comment vas-tu ?"
```

Lors de la saisie n'oubliez pas que vous pouvez corriger vos fautes de frappe en les effaçant avec <Delete> ou <Backspace> et que vous avez la possibilité d'insérer des caractères. Et pour que votre ordinateur fasse réellement ce que vous lui ordonnez, cliquez maintenant sur "Run" dans la barre de menu ou appuyez sur <Shift> <F10> : la barre de menu disparaît car vous vous trouvez dans la fenêtre d'affichage. Ça marche ? Après avoir exécuté l'ordre PRINT, vous voyez apparaître au milieu de votre écran la fenêtre suivante :



Figure 6 : Fenêtre GFA signalant : fin de programme - RETURN



En cliquant sur <RETURN> avec la souris ou en actionnant la touche du même nom vous retournez à la fenêtre de l'éditeur.

Vous tenez certainement à répondre lorsqu'on s'enquiert si gentiment de votre état de santé.

L'Atari doit donc recevoir de votre part un ordre l'incitant à vous *écouter*, ce qui signifie concrètement que votre ordinateur doit se tourner vers le clavier pour percevoir votre réponse : il doit lire quelque chose sur le clavier. Dites le lui donc, mais naturellement en GFA Basic.

L'ordre adéquat est ici INPUT, ce qui signifie approximativement *écoute* ou *lis*, et incite l'ordinateur à *écouter* ce qui provient de son *oreille* : le clavier. Et comme vous êtes intéressé à ce que votre ordinateur n'oublie pas votre réponse, il faut bien qu'il l'enregistre quelque part dans sa mémoire, et il faut même qu'il prenne bonne note de l'endroit où il va enregistrer dans sa mémoire de travail la réponse que vous venez de lui envoyer. C'est pourquoi il a besoin d'un nom ajouté derrière la commande INPUT :

INPUT REPONSE

REPONSE étant ici le nom d'une variable. Si vous vous souvenez plus ou moins confusément de vos cours de mathématiques vous pensez à une autre appellation possible : une *inconnue*. Cela convient tout à fait puisque l'ordinateur ne connaît pas encore votre réponse !

La dénomination d'une variable peut presque prendre n'importe quelle longueur puisque le GFA Basic tolère jusque 255 caractères, acceptant le point (.) ainsi que le tiret de soulignement ( \_ ) en plus des lettres et chiffres. Le premier caractère doit être une lettre. Le GFA Basic distingue tous les caractères d'un nom !

**Attention :** Les caractères accentués (é, è...) ne sont pas autorisés pour les noms de variables.

Vous avez sans doute remarqué que l'éditeur GFA autorise l'écriture en majuscules ou minuscules mais qu'ensuite (dans la version 3.0) il transcrit les commandes Basic en majuscules et que votre mot *REPONSE* est devenu *Reponse* se composant exclusivement de minuscules. Vous pouvez écrire *print* ou *input*, l'éditeur GFA Basic transcrira cela par PRINT ou INPUT. Si cela ne vous convient pas et que vous disposez de la version 3.0, vous retournez au menu GFA-BASIC qui se trouve sous le symbole Atari et vous optez pour un mode d'écriture différent par le biais de *Deflist* (voir annexe B).

Si, par contre, vous disposez de la version 2.0, vous bénéficiez d'une autre présélection, *Deflist 1*, qui fait commencer chaque mot par une majuscule. Vous pouvez ici aussi modifier cette présélection : il vous suffit de vous placer dans le mode *Direct* (en appuyant tout simplement sur la touche <Esc>) puis d'écrire *Deflist* suivi d'un nombre et enfin de revenir dans l'éditeur par <control> <Shift (gauche)> <Alternate>. Dans le présent ouvrage je m'en tiens pour tous les programmes au mode d'écriture engendré par *DEFLIST 0*.

Lancez maintenant votre petit programme à l'aide de *Run*. Tout semble aller pour le mieux, car après un court laps de temps vous voyez s'afficher une gentille phrase sur l'écran de sortie :

```
Bonjour, comment vas-tu ?  
?
```

Après cette phrase, vous voyez le point d'interrogation *INPUT* signifiant que la machine attend votre réponse. Qu'allez-vous répondre ? Admettez que pour l'instant vous allez (encore) plutôt bien. Entrez *bien* et terminez votre réponse en appuyant sur la touche <Return>.

### Que se passe t-il ?

Le point d'interrogation réapparaît ! Cela veut dire que le programme attend toujours de votre part une réponse. votre réponse *bien* a été refusée !

Essayez alors d'entrer le nombre 200. Cette valeur est acceptée. Appuyez sur *Return* pour revenir à l'éditeur.

*REPNSE* contient maintenant la valeur 200. Pour vous en assurer, modifions le programme de façon à afficher le contenu de *REPNSE* :

```
PRINT "Comment vas-tu ?  
INPUT Reponse  
PRINT Reponse
```

Lancez à nouveau ce petit programme en cliquant sur *RUN*.

Entrez la valeur 1000 en réponse au point d'interrogation.

Le résultat suivant apparaît :

```
Comment vas-tu ?  
? 1000  
1000
```

La troisième ligne indique bien que la variable *Reponse* contient la valeur *1000*. Appuyez sur < Return > pour revenir à l'éditeur.

Mais alors, pourquoi l'ordinateur nous a refusé le texte *bien* en réponse au point d'interrogation ? Pourquoi a-t-il accepté qu'on lui réponde *1000* et non *bien*.

Je vous dois bien une explication : tous les noms de variables que vous utilisez en Basic deviennent tout d'abord, comme il est d'usage en mathématiques, des variables numériques. Ceci signifie par exemple qu'elles ne peuvent contenir que des nombres. C'est pourquoi *1000* a été accepté, *bien* a été refusé.

De votre côté cependant, vous ne voulez pas écrire des chiffres mais du texte, c'est-à-dire des caractères, des chaînes de caractères. C'est pourquoi il faut faire figurer dans le programme un signal disant à l'ordinateur que *reponse* contiendra du texte et non un nombre. On ajoute tout simplement ce signal (\$) à chaque variable. Voyez par vous même :

```
PRINT "Bonjour, comment vas-tu ?"  
INPUT reponse$  
PRINT reponse$
```

Et voici enfin votre tout premier programme écrit en GFA Basic : faites le tourner plusieurs fois et entrez des réponses différentes. Cette fois-ci, le texte sera accepté !

## 2.2. Ça va bien, ça va mal ...

Ce programme ne reflète cependant pas encore une véritable conversation ; en effet, votre ST ne tient pas vraiment compte de votre réponse et se borne à une répétition, comme un écho. Vous vous attendez au moins à quelque compréhension de la part de quelqu'un qui s'enquiert de votre état de santé ! Naturellement, vous n'attendez pas de sentimentalité de la part d'un ordinateur, il vous suffit en tant qu'utilisateur d'avoir l'impression que votre ordinateur vous comprend. L'ordinateur peut en effet exploiter les chaînes de caractères de votre réponse et comparer la variable *reponse\$* avec un mot quelconque comme *bien* ou alors *mal* et exécuter ensuite un ordre selon la nature de la réponse reçue :

```
IF reponse$ = "bien"  
  PRINT "je m'en réjouis"  
ENDIF
```

```
IF reponse$ = "mal"
  PRINT "j'en suis bien peiné !"
ENDIF
```

La véritable signification des mots *bien* ou *mal* échappe à votre ordinateur qui se borne à comparer la réponse que vous tapez sur le clavier avec une chaîne de caractères *bien* ou *mal*. Et si (= IF) votre réponse correspond signe pour signe très exactement à la chaîne de caractères, il vous fournit la réponse mémorisée. Dans les chaînes de caractères, l'ordinateur distingue les majuscules des minuscules et tient compte des espaces vides !

Après avoir amené le curseur à la fin du texte de votre programme par <Control>-Z, vous pouvez le compléter. Si vous tentez d'insérer des espaces vides entre

```
reponse$ = "bien"
```

ou

```
reponse$ = "mal"
```

l'éditeur les gommara car il travaille dans un *mode comprimé* et n'accepte pas d'espace vide dans le texte des programmes, sauf si vous les faites précéder d'une apostrophe qui dégage la responsabilité de l'éditeur pour le restant de la ligne. Il réorganise alors les lignes entre IF et ENDIF automatiquement.

Et maintenant faites donc un essai : il vous plaît, votre programme ?

```
PRINT "Bonjour, comment vas-tu ?"
INPUT reponse$
IF reponse$ = "bien"
  PRINT "je m'en réjouis"
ENDIF
IF reponse$ = "mal"
  PRINT "j'en suis bien peiné !"
ENDIF
```

La procédure de contrôle que vous venez d'utiliser a la forme suivante en GFA Basic :

```
IF <condition>
  <instruction>
ENDIF
```

et vous vous doutez bien que plusieurs instructions peuvent figurer entre IF et ENDIF, mais attention le GFA Basic n'admet qu'une instruction par ligne.

Essayez d'écrire sur une seule ligne tout le bloc compris entre IF et ENDIF :  
que vous répond l'éditeur GFA ?

Il est grand temps de sauver sur une disquette votre premier chef d'oeuvre.  
Vous avez deux possibilités :

Save	ou	< Shift > < F1 >
Save,A	ou	< Shift > < F2 >

dans les deux cas vous voyez apparaître une fenêtre de dialogue vous demandant de donner vous-même le nom du fichier sous lequel vous voulez archiver votre oeuvre. Les extensions sont par contre préétablies : GFA ou BAS pour les textes de programmation encodés, LST pour listing qui désigne des fichiers mémorisant tous les mots, y compris ceux du GFA, caractère par caractère.

Quel que soit votre choix, nommez donc votre première tentative *BONJOUR* et terminez en appuyant sur < Return > . Les extensions LST ou GFA (version 3.0) ou BAS (version 2.0) s'inscrivent automatiquement.

### 2.3. Une note plus personnelle ?

Notre petit *bonjour* n'est pas mal pour un début, non ? mais si vous vouliez lui donner une note plus personnelle, vous devriez indiquer votre nom à votre Atari ST, et vous devriez donc l'insérer dans votre programme : c'est d'ailleurs bien mieux lorsque chaque utilisateur se sent apostrophé personnellement. Pour cela, il faut apprendre à l'ordinateur les noms de la ou des personne(s) concernée(s) : vous allez le faire de la même façon qu'avec votre *reponse*. A l'aide de < Control > < ClrHome > , placez-vous dans la fenêtre éditeur en début de programme, où vous allez vous créer une ligne vide par < Ins > sur laquelle vous écrivez :

```
PRINT "Bonjour, comment t'appelles-tu ?"  
INPUT nom$
```

L'éditeur GFA a lui-même libéré une deuxième ligne. Vous devez maintenant modifier les lignes de votre programme en y apportant les corrections nécessaires. Recopiez s'il vous plaît très exactement la ligne en faisant bien attention au point-virgule après *nom\$*. Je vous expliquerai juste après le rôle de ce point-virgule. Descendez tout simplement le curseur d'une ligne, et voilà que l'éditeur a relié les nouvelles lignes avec les anciennes. Puisque vous en

êtes aux modifications, ne serait-il pas plus esthétique et plus simple de donner un vrai nom à votre programme ainsi qu'une terminaison convenable :

```
REM une petite conversation (1)
PRINT "Bonjour, comment t'appelles-tu ?"
INPUT nom$
PRINT nom$;"",comment vas-tu ?"
INPUT reponse$
IF reponse$="bien"
    PRINT "je m'en réjouis"
ENDIF
IF reponse$="mal"
    PRINT "j'en suis bien peiné !"
ENDIF
END
```

Les caractères REM donnent à l'ordinateur l'ordre de ne plus tenir compte de la ligne. REM est l'abréviation de REMark et signifie : je vais maintenant écrire une remarque, une explication, des mots qui ne concernent que le programmeur et pas l'ordinateur. A la place de REM vous pouvez utiliser une apostrophe (').

Cette ligne n'a aucune importance lorsque le programme tourne, mais pour vous en tant que programmeur, quelques remarques disséminées de-ci de-là peuvent être une aide précieuse pour vous rappeler par la suite ce que vous aviez l'intention de faire avec telle ou telle partie du programme.

END vous permet de mettre un point final à un programme : toutes les variables disparaissent, tous les fichiers ouverts par votre programme sont refermés. Vous pouvez bien sûr négliger cette instruction, car lorsque l'ordinateur ne reçoit plus aucune instruction, il considère de toute façon le programme terminé. Mais il n'est jamais inutile de terminer proprement un travail. Dans notre programme,

REM	serait donc la marque du début
END	celle de la conclusion.

Comme nous en sommes aux travaux d'embellissement, je vous propose encore le programme suivant. Si vous avez la version 2.0, n'essayez pas de saisir le programme, passez directement à son commentaire.

```
REM une petite conversation (2)
PRINT "bonjour, comment t'appelles-tu ?"
INPUT nom$
PRINT nom$;"",comment vas-tu ?"
INPUT reponse$
IF reponse$="bien"
```



```
PRINT "je m'en réjouis"
ELSEIF reponse$="mal"
  PRINT "j'en suis bien peiné !"
ELSE
  PRINT "ah bon..."
ENDIF
END
```

Que pensez-vous maintenant de notre programme ? Vous trouvez que c'est une question de goût ? C'est un fait que vous venez de faire connaissance avec une nouvelle instruction du GFA Basic. Et si vous avez fait tourner plusieurs fois ce programme, vous avez sans doute compris la signification de ELSE et ELSE IF :

L'instruction IF..ELSE..ENDIF branche le programme sur différents blocs programmes, suivant la valeur logique (vrai ou faux) de "condition".

L'instruction ELSEIF n'existe que dans la version 3.0. Elle permet de représenter des instructions IF imbriquées de façon plus claire.

ELSE signifie en français *autrement* et plus précisément ici *sinon*. Et maintenant relisez et comparez avec les programmes écrits auparavant. La commande ELSEIF n'existe malheureusement que depuis la version 3.0 ; dans la version 2.0 vous obtenez le même effet en plaçant IF à la ligne suivante et en le faisant suivre de ENDIF :

```
REM une petite conversation (2) GFA V2.0
PRINT "bonjour, comment t'appelles-tu ?"
INPUT nom$
PRINT nom$;"comment vas-tu ?"
INPUT reponse$
IF reponse$="bien"
  PRINT "je m'en réjouis"
ELSE
  IF reponse$="mal"
    PRINT "j'en suis bien peiné !"
  ELSE
    PRINT "ah bon..."
  ENDIF
ENDIF
END
```

Nous aurions pu écrire :

```
IF reponse$="bien"
  PRINT "je m'en réjouis"
ENDIF
IF reponse$ <> "bien"
```

```
PRINT "j'en suis bien peiné !"
ENDIF
```

ou encore :

```
IF reponse$ = "bien"
  PRINT "je m'en réjouis"
ELSE
  PRINT "j'en suis bien peiné !"
ENDIF
```

Mais qu'arrive-t-il si par exemple vous répondez en écrivant *ça va merveilleusement bien* ? Cette fois c'est l'ordinateur qui est peiné ! Cela ne vous plaît pas ? Vous avez une autre possibilité :

```
IF reponse$ = "bien"
  PRINT "je m'en réjouis"
ELSE
  PRINT "ah bon..."
ENDIF
```

Vous avez maintenant la possibilité de reconstruire votre programme en faisant appel aux dernières commandes apprises, et en ayant recours aussi souvent que nécessaire à ELSE IF pour exclure des réponses éventuelles qui ne seraient pas formulées. Pensez à utiliser <Ins> pour intercaler une nouvelle ligne et <Control> <Delete> pour en effacer une que vous pouvez d'ailleurs rappeler par <Control> -U (dans la version 3.0) :

```
:
IF reponse$ = "bien"
  PRINT "je m'en réjouis"
ELSE IF reponse$ = "mal"
  PRINT "j'en suis bien peiné !"
ELSE IF reponse$ = "pas trop bien"
  PRINT "ça ira mieux demain !"
ELSE IF reponse$ = "très très bien"
  PRINT "magnifique !"
:
ELSE
  PRINT "ah bon..."
ENDIF
:
```

Voyez par vous-même comment vous avez progressé. Il ne vous reste plus naturellement qu'à composer des textes qui vous sembleront plus intelligents surtout après PRINT. Et n'oubliez pas de conclure un bloc IF par ENDIF, et non pas END IF !

Pour terminer, vous devriez sauvegarder votre programme. Vous vous souvenez ? avec Save ou les touches <Shift> <F1>. Ou alors avec Save,A ou <Shift> <F2>. Parvenu presque à la fin de ce chapitre, je voudrais ajouter un listing en Basic Standard pour ceux qui pratiquaient par exemple le C64 ou le Basic livré avec l'Atari ST :

```
10 REM une petite conversation
20 PRINT "bonjour, comment t'appelles-tu ?"
30 INPUT N$
40 PRINT N$,"comment vas-tu ?"
50 INPUT R$
60 IF R$="bien" THEN PRINT "je m'en réjouis"
70 IF R$ < > "bien" THEN PRINT "j'en suis tout triste"
70 END
```

Vous pouvez aussi utiliser THEN dans le GFA Basic version 3.0 ce qui ferait :

```
IF reponse$="bien"
IF reponse$="bien" THEN
```

Mais comme THEN n'a plus d'autre signification dans le GFA Basic, nous ne l'utiliserons plus dans les programmes suivants. Le GFA Basic n'autorise pas la numérotation des lignes, qui est d'ailleurs inutile.

## 2.4. En résumé

Vous connaissez maintenant les mots suivants dans le vocabulaire du GFA Basic :

```
PRINT
INPUT
IF (..THEN)
ELSE
ELSE IF
ENDIF
REM (ou ')
END
```

J'espère que vous vous souvenez de leur signification. En tout cas, vous connaissez une instruction d'affichage :

```
PRINT
```

et une instruction de saisie :

```
INPUT
```

Vous savez que le Basic fait la différence entre les chiffres et les lettres, et reconnaît une variable texte grâce au suffixe "\$".

De plus, vous connaissez une instruction qui teste la correspondance exacte entre deux chaînes de caractères :

```
IF {condition}
  {instruction}
ELSE
  {autre instruction}
ENDIF
```

Vous vous souvenez même qu'avec le GFA Basic vous pouvez emboîter ces instructions de contrôle grâce à :

```
ELSE IF (version 3.0)
```

ou :

```
ELSE
IF
ENDIF (version 2.0)
```

Et vous n'oubliez pas de terminer vos blocs commençant par IF avec ENDIF !

Vous savez que le GFA Basic admet des noms de variables d'une longueur pouvant aller jusqu'à 255 caractères et qu'il distinguera tous les caractères.

Si vous voulez donner un nom à un programme ou insérer des remarques dans un texte de programme, vous devez utiliser

```
REM ou apostrophe (')
```

ajoutez à cela END et vous avez la possibilité de marquer le début et la fin d'un programme. D'autre part :

Save	ou	< Shift > < F1 >	(.GFA) (.BAS)
Save,A	ou	< Shift > < F2 >	(.LST)

vous permettent de sauvegarder vos textes dans un fichier dont vous choisissez le nom et

Load	ou	< F1 >	(.GFA) (.BAS)
Merge	ou	< F2 >	(.LST)

vous permettent soit de les charger soit de les ajouter. La commande :

Run	ou	< Shift > < F10 >
-----	----	-------------------

vous permet de lancer votre programme.

Pour effacer un programme dans l'éditeur vous allez utiliser

New	ou	< Shift > < F4 >
-----	----	------------------

et pour quitter définitivement l'éditeur GFA vous faites

Quit	ou	< Shift > < F3 >
------	----	------------------



## Chapitre 3

### Une autre perception du Basic

Maintenant que vous avez réussi à écrire votre premier programme en GFA Basic, il est temps de vous accorder une pause pour souffler avant de vous plonger plus avant dans l'étude du GFA Basic. Laissez-moi vous parler un peu du Basic en général et de ce dialecte Basic en particulier.

#### 3.1 Pourquoi justement un GFA Basic ?

Que savez-vous du Basic ? Qu'il s'agit certes d'un langage multi-usages destiné surtout aux débutants, facile à apprendre et à utiliser. Que ce langage a(vait) la réputation de voir les problèmes croître en même temps que la complexité des programmes, que sa simplicité pouvait vite conduire à un manque de clarté lors de l'écriture de programmes longs et complexes. Cette image du Basic amena la désaffection des programmeurs professionnels à son égard, entraîna d'autre part l'apparition d'une émulation entre programmeurs obtenant des résultats étonnants grâce au Basic mais ceci le plus souvent grâce à des astuces de programmation incompréhensibles.

Car dans sa forme *primitive* le Basic n'était pas très performant en ce qui concerne l'articulation et la clarté. Ceux parmi vous qui s'y connaissent devront m'accorder qu'avec une poignée de commandes -PRINT, INPUT, LET, IF, THEN et GOTO - ainsi que quelques signes (= < > ") et l'emploi de la numérotation des lignes, on peut écrire une foule de programmes. Par de nombreuses combinaisons plus ou moins heureuses de ces éléments, on peut tricoter de grands ouvrages qu'on ne parvient quasiment plus à démêler ensuite. Voilà ce qui autrefois a empêché l'utilisation du Basic pour de grands projets.

Il serait bien sûr exagéré d'affirmer que tout cela a changé avec le GFA Basic. D'abord parce qu'il existe d'autres variantes du Basic pour l'Atari qui prennent en compte ce besoin de clarté dans l'élaboration du programme, et ensuite parce que même avec le GFA Basic un projet de construction peut devenir un labyrinthe si les possibilités puissantes de ce Basic sont mal utilisées.

Qu'attend-on d'un bon langage de programmation ? Il devrait être facile à apprendre et très proche de notre langue usuelle. Si on fait abstraction du fait que ce langage ne dispose pas d'un vocabulaire français mais anglais, cette condition est remplie par presque toutes les versions du Basic. Mais ce n'est pas tout, car on voudrait de plus :

- ✱ pouvoir manipuler aisément les données
- ✱ disposer de structures de contrôle efficaces pour le découpage et la répétition de séquences dans les programmes, si possible même avoir le choix entre plusieurs à la fois pour s'adapter au mieux à la situation donnée par la programmation en cours
- ✱ avoir la possibilité de découper des programmes longs en blocs plus facilement maîtrisables et de convenir de quelques routines et procédures.

Le Basic-standard offert dans le monde des micro-ordinateurs comme par exemple le C64, offre peu de possibilités dans tous ces domaines. Les versions jointes aux Amiga, Macintosh ou PC sont déjà plus satisfaisantes.

Les autres langages comme le Pascal, C et Modula offrent une foule de possibilités pour structurer les programmes. Et le GFA Basic n'a pas à rougir devant eux comme vous allez pouvoir le constater.

## 3.2. Interpréteur ou compilateur ?

Dans la plupart des cas, le Basic est un langage Interprété : les lignes d'instruction sont exécutées une par une, et traduites sous une forme compréhensible pour l'ordinateur ; lorsqu'il rencontre une faute, il peut tout de même exécuter le programme jusqu'à l'endroit défectueux. Qui plus est, un interpréteur exécute immédiatement les instructions données en mode direct.

Le GFA Basic est lui aussi un interpréteur. Mais il appartient aux interpréteurs ultra-rapides, surtout si on le compare aux interpréteurs-Basic d'autres ordinateurs. Toutefois, si sa rapidité ne vous suffisait pas, GFA vous offre



également un compilateur approprié (pour la version 2.0 actuellement), qui traduit toujours en bloc l'ensemble du programme en langage machine, et ne le met en route qu'après avoir vérifié l'absence de faute. Un compilateur ne peut pas exécuter des ordres entrés en mode direct.

A chaque *Run*, l'interpréteur relance le programme en l'exécutant ligne par ligne, et il le retraduit à chaque fois. Ceci prend beaucoup de temps, or aucun programme ne peut tourner sans son interpréteur. Par contre un compilateur, après avoir traduit une fois un programme sans faute et y avoir ajouté ce qu'on appelle un module de routines, rend ce programme définitivement utilisable et compréhensible pour l'ordinateur sans son compilateur.

Un programme compilé tient peu de place dans la mémoire, travaille rapidement et de façon autonome. Un programme interprété reste lent et nécessite toujours son interpréteur. Les autres versions du Basic doivent de surcroît toujours traîner avec elles leur éditeur, alors que le GFA Basic vous offre l'interpréteur *Run-Only* seul (GFABASRO.PRG), ce qui vous permet de charger et faire tourner automatiquement des programmes finis écrits en Basic. Mais si vous voulez ensuite les modifier, vous devez recourir à la version avec éditeur intégré.

Pour fabriquer des programmes insurpassables en rapidité et en densité, vous devriez parler à votre ordinateur ST directement dans sa propre langue : le langage machine. Son alphabet ne comprend que les signes 1 et 0, que l'on aligne dans l'ordre voulu. Ce nombre si *important* de signes rend les chaînes d'écriture longues et indéchiffrables.

Cette forme de langage peut très facilement nous faire commettre des erreurs, et nous faire dire à l'ordinateur des choses que nous n'aurions jamais dû lui dire !

Pour que vos conversations ne soient pas trop sèches et pénibles, vous disposez du langage assembleur : les chaînes de zéros et de uns sont remplacées par des abréviations d'instructions facilement mémorisables. Ce type de programmation n'est pas seulement plus difficile qu'avec le Basic, mais peut être la source d'une grande confusion. Ne vous y risquez donc pas, et servez-vous plutôt d'un interprète ou d'un traducteur pour donner des instructions à votre ordinateur.

### 3.3. To GO or not to GO...

En choisissant de débiter avec le GFA Basic, vous avez la possibilité de passer directement à la programmation structurée. En adoptant cette nouvelle version du Basic après avoir utilisé les anciennes, vous avez l'occasion de vous débarrasser des mauvaises habitudes de la *programmation-spaghetti*. Par exemple, vous n'aurez plus à utiliser la numérotation des lignes existante dans le Basic-standard. Pour gérer des boucles, il suffit de placer des "marques" (Label). Chose impossible avec le basic-standard : grâce au GFA Basic vous pouvez très largement vous passer du GOTO, car vous aurez de nombreuses autres possibilités.

Je voudrais pour les exemples suivants faire appel de nouveau à notre petit programme *'bonjour'*, en le modifiant quelque peu. Le premier exemple est écrit en style GOTO tout à fait classique pour un interpréteur du Basic standard, le deuxième exemple représente une solution intermédiaire, et le troisième ce qu'on appelle une version *NOGOTO* :

```
5 REM un bonjour bouclé (1)
10 PRINT "bonjour, comment vas-tu ?"
20 INPUT reponse$
30 IF reponse$ = "bien" THEN GOTO 100
40 IF reponse$ = "mal" THEN GOTO 110
50 PRINT "tu vas bien ou tu vas mal ?" : GOTO 20
100 PRINT "je m'en réjouis" : GOTO 120
110 PRINT "j'en suis bien peiné !"
120 END
```

```
REM un bonjour bouclé (2)
PRINT "bonjour, comment vas-tu ?"
saisie :
INPUT reponse$
IF reponse$ = "bien"
  GOTO positif
ENDIF
IF reponse$ = "mal"
  GOTO negatif
ENDIF
PRINT "tu vas bien ou tu vas mal ?"
GOTO saisie
positif :
PRINT "je m'en réjouis"
END
negatif :
PRINT "j'en suis bien peiné !"
END
```

## Version 3.0

```

REM un bonjour sans boucle
PRINT "bonjour, comment vas-tu ?"
REPEAT
  INPUT reponse$
  IF reponse$="bien"
    PRINT "je m'en réjouis"
  ELSEIF reponse$="mal"
    PRINT "j'en suis bien peiné !"
  ELSE
    PRINT "tu vas bien ou tu vas mal ?"
  ENDIF
UNTIL reponse$="bien"
  OR reponse$="mal"
END

```

## Version 2.0

```

REM un bonjour sans boucle (3)
PRINT "bonjour, comment vas-tu ?"
REPEAT
  INPUT reponse$
  IF reponse$="bien"
    PRINT "je m'en réjouis"
  ELSE
    IF reponse$="mal"
      PRINT "j'en suis bien peiné"
    ELSE
      PRINT "tu vas bien ou tu vas mal"
    ENDIF
  ENDIF
UNTIL reponse$="bien"
  OR reponse$="mal"
END

```

Et maintenant ne me reprochez pas d'avoir utilisé trop de GOTO dans les deux premières versions, et ne me dites pas qu'il est possible d'écrire ce programme de façon beaucoup plus élégante sans le GFA Basic, car vous me couperiez l'herbe sous les pieds. En effet, plus mes exemples seront rebutants, plus vous serez disposés à faire autrement.

Tournons-nous vers le troisième exemple : peut-on reconnaître la structure de ce programme ? Avez-vous bien constaté que REPEAT et UNTIL encadrent des instructions qui doivent être répétées jusqu'à ce qu'on réponde par *bien* ou *mal* ?

L'instruction REPEAT...UNTIL, sert à exécuter un bloc d'instructions jusqu'à ce qu'une condition soit remplie. Une boucle REPEAT...UNTIL est donc forcément parcourue au moins une fois.

Et maintenant représentez-vous chacun de ces trois exemples en tant qu'extrait d'un programme de plusieurs centaines de lignes : dans le premier cas, vous chercherez vainement après une structure, car il n'y en a pas. Dans le deuxième cas, on soupçonne l'intention d'une structuration. Le troisième cas vous permet de percevoir clairement l'articulation du programme en blocs distincts.

Le style *cross-country* paraît le plus simple : vous courez tête baissée, si vous vous trompez de chemin, vous changez tout simplement de direction ou revenez le cas échéant à l'embranchement voulu (GOTO). Ceci marche bien pour de brefs circuits. Mais si vous avez par contre un grand parcours devant vous, il n'est pas sûr que vous arriviez au but !

Une *programmation structurée* nécessite par contre une plus grande préparation, mais vous serez bien plus certain d'arriver au but à temps. Alors que d'autres langages de programmation comme le Pascal vous obligent à un tel style, le Basic vous laisse le choix et la possibilité de programmer sans but précis. Le GFA Basic vous offre un peu des deux : un peu de la liberté qui fait apprécier le Basic, et beaucoup des possibilités des langages disciplinés comme le C ou Pascal.

### 3.4. Pour votre confort

Si mes explications théoriques vous ont quelque peu ennuyé, voici de nouveau un peu de pratique. Je voudrais compléter votre savoir de débutant par quelques subtilités propres au GFA Basic. Reprenez votre première oeuvre :

#### Version 3.0

```
REM une petite conversation (1)
PRINT "bonjour, comment t'appelles-tu ?"
INPUT nom$
PRINT nom$;" comment vas-tu ?"
INPUT reponse$
IF reponse$="bien"
  PRINT "je m'en réjouis"
ELSEIF reponse$="mal"
  PRINT "j'en suis bien peiné !"
ELSE
  PRINT "ah bon..."
ENDIF
END
```

#### Version 2.0

```
REM une petite conversation
PRINT "bonjour, comment t'appelles-tu ?"
INPUT nom$
PRINT nom$;" comment vas-tu ?"
INPUT reponse$
IF reponse$="bien"
  PRINT "je m'en réjouis"
ELSE
  IF reponse$="mal"
    PRINT "j'en suis bien peiné"
  ELSE
    PRINT "ah bon..."
  ENDIF
ENDIF
ENDIF
END
```

Laissez-moi bricoler un peu ce programme :

### Version 3.0

```
REM une petite conversation (2)
INPUT "bonjour, comment t'appelles-tu
?";nom$
PRINT
PRINT nom$;"comment vas-tu ?"
INPUT """,reponse$
PRINT
IF reponse$="bien"
  PRINT "je m'en réjouis"
ELSE IF reponse$="mal"
  PRINT "j'en suis bien peiné !"
ELSE
  PRINT "ah bon..."
ENDIF
END
```

### Version 2.0

```
REM une petite conversation (2)
INPUT "bonjour, comment t'appelles-tu
?"";nom$
PRINT
PRINT nom$;"comment vas-tu ?"
IF reponse$="bien"
  PRINT "je m'en réjouis"
ELSE
  IF reponse$="mal"
    PRINT "j'en suis bien peiné !"
  ELSE
    PRINT "ah bon..."
  ENDIF
ENDIF
ENDIF
END
```

Essayez cette version : vous savez encore comment appeler l'éditeur et l'interpréteur. *Load* ou <F1> ou encore *Merge* ou <F2> vous permettent de charger l'ancien programme depuis le fichier, de le modifier et ensuite de le lancer par *Run* ou <Shift> <F10>. Vous pouvez ensuite, si vous le désirez, sauvegarder la nouvelle version grâce à *Save* ou <Shift> <F1> ou encore *Save,A* ou <Shift> <F2>.

Qu'est-ce-qui change ici ? *PRINT* écrit seul engendre une ligne vide. Mais surtout, *PRINT* déclenche un saut de ligne. S'il faut continuer directement après cette sortie, *PRINT* doit être suivi d'une petite indication, le point virgule (;) qui pourrait être remplacé par une virgule (,) ou une apostrophe ('). Vous verrez par vous même que l'effet n'est pas alors tout à fait le même.

Comme vous le constatez, on peut remplacer *PRINT texte* suivi de *INPUT variable* par *INPUT texte ; variable*. On peut de même lors de la saisie laisser tomber le point d'interrogation grâce à la virgule (,) employée avec *INPUT* entre le texte et la variable.

Sachez finalement que vous pouvez écrire les instructions que vous connaissez de façon abrégée, l'éditeur s'y retrouvera :

p	remplace PRINT
inp	remplace INPUT
i	remplace IF
el	remplace ELSE (ou ELSE IF)
en	remplace ENDIF
r (ou ')	remplace REM

Ces abréviations ainsi que toutes celles du vocabulaire GFA Basic sont récapitulées et classées pour en faciliter l'accès dans le chapitre 16.

### 3.5. Résumé

Et voilà que vous en savez de nouveau un petit peu plus : par exemple vous connaissez la différence entre un Interpréteur et un Compilateur.

L'interpréteur traduit le programme en train de tourner ligne par ligne en langage machine ; le programme ne peut tourner sans son interpréteur.

Le compilateur traduit d'abord l'ensemble du programme en langage machine ; le programme machine peut ensuite tourner sans le compilateur.

Vous savez aussi que PRINT écrit seul engendre un saut de ligne et que :

```
PRINT "texte"
```

suivi de

```
INPUT <variable >
```

peut être remplacé par

```
INPUT "texte" ; <variable >
```

Si vous le désirez, vous pouvez remplacer le point d'interrogation par

```
INPUT "", <variable >
```

et pour finir, essayez de vous souvenir des possibilités d'abréviation des instructions.

## **Partie 2**

**Le savoir de base en**

**GFA Basic**





## Chapitre 4

### Les sauts et les boucles

Peut-être vous est-il déjà arrivé après avoir acheté un nouveau disque de votre interprète favori, de vous asseoir dans un fauteuil pour l'écouter. A un certain moment, vous bondissez en entendant :

```
:  
"you are"  
"you are"  
"you are"  
:
```

Une répétition bien mal venue engendrée par une rayure. Et même une répétition sans fin que vous devez interrompre vous même, ce qui vous irrite car voilà votre disque fichu.

Vous pensez que ma petite histoire n'a pas sa place ici ? Imaginez-vous l'humeur dans laquelle peut vous mettre un programme en Basic. L'habitude très répandue de s'asseoir devant son ordinateur et de se mettre à programmer tête baissée rend l'apparition de telles boucles quasiment inévitable. Et lorsqu'après une séance de programmation ardue on veut récolter les fruits de son activité, on n'est pas à l'abri de ces mauvaises rencontres que sont par exemple des boucles sans fin. Et nous voilà en plein dans notre sujet.

## 4.1. Les sauts sont possibles...

J'espère que vous ne me tiendrez pas rigueur d'avoir encore une fois recours à une variante de notre petit programme Bonjour à titre d'exemple concret.

L'ordinateur doit d'abord s'enquérir du nom de son partenaire. Au lieu de demander *comment vas-tu ?* il doit cette fois demander *te portes-tu bien ?* question à laquelle il suffit de répondre par *oui* ou *non*. Ceci simplifie notre comparaison, réduite à deux possibilités. Si vous n'aviez qu'un Basic standard à votre disposition, votre programme ressemblerait à ce qui suit :

```
10 REM Une petite conversation (1)
20 INPUT "bonjour, comment t'appelles-tu ?" ;n$
30 PRINT n$ ; ",te portes-tu bien ?"
40 INPUT r$
50 IF r$="o" THEN PRINT "je m'en réjouis"
60 IF r$="n" THEN PRINT "j'en suis bien peiné"
70 END
```

Comme vous le savez depuis le chapitre 3, vous pouvez intégrer des textes dans l'instruction INPUT. Le point-virgule derrière le PRINT de la ligne 30 exige que votre réponse soit entrée juste après la question. Pour simplifier, cette réponse peut se réduire aux premières lettres de *oui* et *non*, ce qui est tout à fait habituel dans beaucoup de logiciels - comme vous l'avez peut-être déjà constaté.

En GFA Basic, la version ci-dessus ne pourrait pas tourner, c'est pourquoi nous allons de nouveau considérer la version modifiée en conséquence :

```
REM Une petite conversation (2)
INPUT "bonjour, comment t'appelles-tu ?";nom$
PRINT nom$ ; ",te portes-tu bien ?"
INPUT reponse$
IF reponse$="o"
  PRINT "je m'en réjouis"
ENDIF
IF reponse$="n"
  PRINT "j'en suis bien peiné"
ENDIF
END
```

Remarquez comme j'ai bien *intégré* le point d'interrogation de saisie des données dans la conversation ? Cela ne vous plaît-il pas ?

A la place du deuxième IF, nous aurions pu utiliser beaucoup plus élégamment ELSE :

```

:
INPUT reponse$
IF reponse$ = "o"
  PRINT "je m'en réjouis"
ELSE
  PRINT "j'en suis bien peiné"
ENDIF
:

```

Vous croyez que ELSE est ici un équivalent absolu ? Pensez donc que le clavier de votre ST possède une multitude de touches et qu'il va répondre *j'en suis bien peiné* pour toutes les touches qui ne sont pas "o" Comment cela se passerait-il avec ELSE IF ? Essayez donc !

Il nous faudrait maintenant modifier le programme de telle sorte qu'il ne soit possible d'utiliser que les touches "o" ou "n". Si ce n'est que cela !" dirait un utilisateur averti du Basic standard et il écrirait ainsi le premier exemple :

```

10 REM Une petite conversation (3)
20 INPUT "bonjour, comment t'appelles-tu ?";n$
30 PRINT n$;" ,te portes-tu bien ?"
40 INPUT r$
45 IF r$ < > "o" AND r$ < > "n" THEN GOTO 40
50 IF r$ = "o" THEN PRINT "je m'en réjouis"
60 IF r$ = "n" THEN PRINT "j'en suis bien peiné"
70 END

```

En GFA Basic ceci ne fait aucune difficulté : au lieu d'un numéro de ligne c'est une étiquette (label) qui délimite ici le saut à exécuter :

```

REM une petite conversation (4)
PRINT "bonjour, comment t'appelles-tu ?";nom$
PRINT nom$;" ,comment vas-tu ?"
saisie :
INPUT reponse$
IF reponse$ < > "o" AND reponse$ < > "n"
  GOTO saisie
ENDIF
IF reponse$ = "o"
  PRINT "je m'en réjouis"
ELSE
  PRINT "j'en suis bien peiné !"
ENDIF
END

```

L'ordinateur teste d'abord l'instruction *INPUT reponse\$* : si la réponse n'est ni "o" ni "n", il répète la demande, ce qui signifie qu'il a fait un saut pour revenir en arrière dans le programme.

L'instruction *GOTO Saisie* permet au programme de reprendre le traitement des instructions à partir de l'étiquette *saisie* ..

*GOTO* quant à lui contribue à répandre une mauvaise habitude : vous êtes assis devant votre ordinateur et vous n'avez encore qu'une idée confuse de ce que vous voulez faire. Vous espérez bien que la clarté viendra au fur et à mesure de l'écriture du programme. Lorsque vous rencontrez des difficultés, vous les laissez tout simplement de côté.

Si vous pensez que vous n'êtes pas comme ça, essayez donc de faire un petit programme : après avoir introduit un nombre quelconque, vous devez obtenir sa valeur inverse, qui est 1 divisé par ce nombre - vous vous en rappelez ?

Admettons que vous ayez déjà écrit quelques lignes semblables à ceci :

```
INPUT "entrez un nombre : ",nombre
inverse = 1/nombre
PRINT "la valeur inverse de " ;nombre ;" est " ;inverse
```

Soudain vous hésitez, ou vous ne le remarquez qu'après quelques essais : zéro n'a pas de valeur inverse, car on ne peut pas diviser par zéro. Vous réfléchissez un peu et vous pensez sauver la situation par un petit saut après avoir d'abord délimité le passage visé :

```
saisie :
INPUT "entrez un nombre : ",nombre
IF nombre = 0
  GOTO saisie
ENDIF
inverse = 1/nombre
PRINT "la valeur inverse de " ;nombre ;" est " ;inverse
```

## 4.2. ...mais ils ne sont pas indispensables

Voici la situation dans laquelle vous pourriez vous trouver si vous vous mettez à programmer directement sans planifier auparavant vos intentions : à un certain endroit du programme vous remarquez qu'une partie du texte devrait se répéter, car en certaines circonstances il faut que le programme se répète ou saute à une autre instruction.

Il ne vous reste plus alors qu'à rechercher l'endroit où la boucle devrait commencer et où ensuite le programme devrait continuer. Après avoir posé une balise (à l'aide du double-point) vous n'avez plus qu'à y envoyer l'ordinateur par un GOTO.

Peut-être avez-vous réfléchi un peu plus longuement et pensé à toutes les instructions nécessaires ? Vous savez donc avant même d'écrire votre programme, à quel endroit la boucle doit commencer et se terminer.

Le GFA Basic vous offre deux instructions pour découper votre programme :

REPEAT      pour marquer le début de la boucle
--

et

UNTIL      pour marquer la fin de la boucle.
--

Voyons cela de plus près avec l'exemple de la saisie d'un nombre :

: REPEAT INPUT "entrez un nombre :",nombre UNTIL :
--

Pour le calcul d'une valeur inverse, il reste encore une condition à remplir : il faut recommencer la saisie aussi longtemps que nécessaire jusqu'à ce qu'un nombre différent de zéro ( $< > 0$ ) soit entré.

Donc notre programme pourrait avoir l'allure suivante :

REM Valeur inverse d'un nombre REPEAT INPUT "entrez un nombre : ",nombre UNTIL nombre < > 0 inverse = 1/nombre PRINT "la valeur inverse de ";nombre;"est";inverse END
---

Reprenons notre programme Bonjour et essayons d'y appliquer ce que nous venons d'apprendre. Attendez ! essayez d'abord tout seul et ensuite vérifiez :

REM une petite conversation (5) INPUT "bonjour, comment t'appelles-tu ?";nom\$ PRINT nom\$;" ,te portes-tu bien ?"
--

```

REPEAT
  INPUT reponse$
UNTIL reponse$="o" OR reponse$="n"
IF reponse$="o"
  PRINT "je m'en réjouis"
ELSE
  PRINT "j'en suis bien peiné !"
ENDIF
END
    
```

Nous devrions comparer les deux variantes (4) et (5) de notre *conversation*. Que constatons-nous ? Examinez de plus près les deux conditions :

```

(5) UNTIL reponse$="o" OR reponse$="n"
(4) IF reponse$ < > "o" AND reponse$ < > "n"
    GOTO saisie
ENDIF
    
```

Nous remarquons que les lignes :

```

reponse$="o" and "reponse$="n"
reponse$ < > "o" and reponse$ < > "n"
    
```

sont soit fausses, soit vraies. On dit qu'elles renvoient des valeurs logiques (VRAI ou FAUX). GFA Basic soutient justement des variables pour gérer des réponses logiques. Ces variables doivent être suivies du symbole !.

Par exemple, nous allons créer deux variables logiques nommées *rupture !* et *repetition !* :

```

rupture !    = reponse$="o" OR reponse$="n"
repetition ! = reponse$ < > "o" AND reponse$ < > "n"
    
```

Ainsi, supposons que *reponse\$* contienne la valeur "p". La variable *rupture !* retourne une réponse fausse alors que *Repetition !* renvoie une réponse vraie. Dans ce cas, *Rupture !* est une variable valant 0 (*rupture !* = 0), *repetition !* vaut par contre -1 (*repetition !* = -1). Ces variables s'appellent couramment des variables booléennes.

Nous allons donc exploiter cette facilité syntaxique pour clarifier nos programmes.

## Version 2.0 et Version 3.0

```

REM une petite conversation (5)
INPUT "bonjour, comment t'appelles-tu ?";Nom$
PRINT nom$;" , te portes-tu bien ?"
REPEAT
  INPUT reponse$
  RUPTURE ! = reponse$ = "o" or reponse$ = "n"
UNTIL RUPTURE ! = true
  IF reponse$ = "o"
    PRINT "je m'en réjouis"
  ELSE
    PRINT "j'en suis peiné !"
  ENDIF
END

```

La boucle	{ Repeat ..... Until rupture ! = true
-----------	---

est équivalente à	{ Repeat ..... Until rupture ! = -1
-------------------	---

ou	{ Saisie if repetition ! Goto saisie Endif
----	---

Dans les deux cas, une fois parvenu à la fin de la boucle, l'interpréteur vérifie si la condition est bien remplie :

Dans la *conversation 4* la boucle est répétée par un saut lorsque la condition *repetition* est remplie, sinon la boucle est terminée.

Dans la *conversation 5* la boucle prend fin lorsque la condition *rupture* est remplie, sinon elle se répète.

Si les deux versions du programme ont bien le même effet, l'une des conditions représente cependant le contraire de l'autre. Il en va ainsi puisque UNTIL sert bien à mettre fin à la boucle alors que GOTO sert à répéter la boucle.

Il apparaît donc que chaque proposition peut se transformer en son contraire ; ici par exemple "=" devient "< >" et "OR" devient "AND".

On peut sans doute discuter pour savoir quelle version paraît la plus logique. Je dis à l'ordinateur d'attendre une certaine réponse. Si elle ne remplit pas

certaines conditions, il doit répéter sa demande et retourner à l'instruction de saisie. Ou alors je dis à l'ordinateur qu'il doit laisser entrer des réponses jusqu'à ce que l'une d'elles remplisse les conditions définies.

### 4.3. Aussi longtemps...jusqu'à ce que...

Nous n'avons parlé jusqu'ici que de la fin de la boucle. Mais toute boucle a un début ! et c'est cela que nous allons maintenant examiner :

```
REM une petite conversation (6)
INPUT "bonjour, comment t'appelles-tu ?";nom$
PRINT nom$;" te portes-tu bien ?"
WHILE reponse$ < > "o" AND reponse$ = < > "n"
  INPUT reponse$
WEND
IF reponse$ = "o"
  PRINT "je m'en réjouis"
ELSE
  PRINT "j'en suis bien peiné !"
ENDIF
END
```

Voici le même programme utilisant la variable booléenne *repetition* ! :

```
REM petite conversation (6)
INPUT "bonjour, comment t'appelles-tu ?";nom$
PRINT nom$;" te portes-tu bien ?"
WHILE repetition ! = False
  INPUT reponse$
  Repetition ! = Reponse$ < > "o" and reponse$ < > "n"
  IF reponse$ = "o"
    PRINT "je m'en réjouis"
  ELSE
    PRINT "j'en suis peiné"
  ENDIF
WEND
END
```

Voilà qui rend possible un contrôle dès le début de la boucle ! Ce qui ne change rien à la condition requise. Etudions de nouveau les limites de la boucle :

```
WHILE repetition !
{
WEND
```



La nouvelle version du GFA Basic vous offre d'ailleurs encore d'autres possibilités que je ne vais pas vous cacher :

```
REM une petite conversation (7)
INPUT "bonjour, comment t'appelles-tu ?";nom$
PRINT nom$; ", te portes-tu bien ?"
DO
  INPUT reponse$
LOOP UNTIL reponse$ = "o" OR reponse$ = "n"
IF reponse$ = "o"
  PRINT "je m'en réjouis"
ELSE
  PRINT "j'en suis bien peiné !"
ENDIF
END
```

### Résumons :

L'instruction DO LOOP effectue une boucle de traitement de la même façon que REPEAT UNTIL. La nuance est dans le fait que la boucle est sans fin à moins d'y intégrer l'instruction EXIT IF.

L'instruction WHILE WEND crée une boucle contenant un bloc d'instructions. Ces instructions sont traitées SI la condition est remplie et tant que la condition est remplie, cela veut dire qu'un premier parcours n'est autorisé que si la condition est vérifiée.

### Version 3.0

```
REM une petite conversation (8)
INPUT "bonjour, comment t'appelles-tu ?";nom$
PRINT nom$; ", te portes-tu bien ?"
DO WHILE reponse$ < > "o" AND reponse$ < > "n"
  INPUT reponse$
LOOP
IF reponse$ = "o"
  PRINT "je m'en réjouis"
ELSE
  PRINT "j'en suis bien peiné !"
ENDIF
END
```

Les boucles ont la forme suivante :

```
DO
{
LOOP UNTIL rupture !
```

ou encore :

```
DO WHILE repetition !  
{  
  LOOP
```

Et si vous essayez, vous constaterez que vous pouvez aussi combiner DO UNTIL et LOOP WHILE, mais attention, seulement dans la version 3.0.

A y regarder de plus près, les appellations *repetition !* et *rupture !* ne sont pas toujours bien appropriées. Car il n'est pas sans importance de placer la condition entraînant l'exécution de la boucle au début ou à la fin de cette dernière.

Dans le cas de WHILE...WEND, DO WHILE...LOOP ou DO UNTIL...LOOP, les valeurs des réponses sont testées avant d'entrer dans la boucle. Dans certains cas, il n'est donc même pas besoin d'exécuter celle-ci. Ce type de boucle est recommandé pour repérer des valeurs fausses qui pourraient faire échouer le programme lorsqu'il tourne.

Dans le cas de REPEAT...UNTIL, DO...LOOP WHILE ou DO...LOOP UNTIL, la boucle est parcourue au moins une fois, car l'ordinateur doit parcourir la boucle pour tester les valeurs.

Ce type de boucle est recommandé pour tester des conditions qui apparaissent dans la boucle ou qui se transforment.

### Exercice pour la version 3.0 :

Comme vous le constatez, le GFA Basic (version 3.0) vous offre de nombreuses possibilités de boucles. A ceux qui ont donc cette version, je leur demande de bien vouloir pratiquer l'exercice suivant. Entrez ce programme mais ne l'exécutez pas avant d'avoir lu la suite du texte :

```
REM un grand bonjour  
antique = 100  
eternite = 100  
INPUT "bonjour, comment t'appelles-tu ?";nom$  
PRINT nom$;" ,te portes-tu bien ? (o/n)";  
DO  
  INPUT reponse$  
LOOP  
IF reponse$ = "o"  
  PRINT "je m'en réjouis"  
ELSE  
  PRINT "j'espère que ça va s'améliorer !"
```

```

ENDIF
PRINT
PRINT "quel âge as-tu donc ?" ;
DO
  INPUT age
LOOP
IF age > 2 AND age < 20
  PRINT "donc tu sais déjà marcher"
ELSE IF age >= 20
  PRINT "donc tu es un grand"
ENDIF
pro = age*100/antique
PRINT "et tu as déjà";pro;"% de ta vie derrière toi !"
PRINT
beaucoup = eternite-age
PRINT "je te souhaite encore ";beaucoup;" d'années à vivre, ";nom$;" !"
END

```

Si vous tenez à faire tourner le programme "un grand bonjour" tel qu'il est écrit ci-dessus, sachez que vous pouvez interrompre le déroulement d'un programme en appuyant simultanément les touches <Alternate> <Shift> <Control>...

Vous trouvez dans ce programme deux boucles DO...LOOP sans condition. Dans la version 3.0, complétez-les de façon à ce que dans le premier cas seules les réponses "o" ou "n" soient possibles et que dans le deuxième cas on ne puisse entrer qu'un âge compris entre 1 et 99 ans. Prenez le temps de faire tourner toutes les versions possibles.

Voici un exemple de solution pour la version 3.0 :

```

REM un grand bonjour
antique = 100
eternite = 100
INPUT "bonjour, comment t'appelles-tu ?",nom$
PRINT nom$;" , te portes-tu bien ?"
DO
  INPUT reponse$
  LOOP UNTIL reponse$ = "o" OR reponse$ = "n"
  IF r$ = "o"
    PRINT "je m'en réjouis"
  ELSE
    PRINT "j'espère que ça va s'améliorer !"
  ENDIF
PRINT
PRINT "quel âge as-tu donc ?";
DO
  INPUT age
  LOOP UNTIL age > 1 AND age < 99
  IF age > 2 AND age < 20

```

```
PRINT "donc tu sais déjà marcher"
ELSE IF age >= 20
PRINT "donc tu es grand"
ENDIF
pro = age*100/antique
PRINT "et tu as déjà ";pro;" de ta vie derrière toi"
PRINT
beaucoup = eternite-age
PRINT "je te souhaite encore ";beaucoup;" d'années à vivre ";nom$;" !"
END
```

### Exercice pour la version 2.0 :

Nous allons maintenant proposer le même exercice, mais adapté à la version 2.0. Dans cet exercice, le programme contiendra deux boucles DO...LOOP. Cependant, ce type de structure est une boucle sans fin, c'est-à-dire qu'elle ne s'arrête pas. Pour interrompre ce genre de situation, il faut utiliser une instruction EXIT IF < condition > . Prenons l'exemple suivant :

```
DO
INPUT nom$
LOOP
```

C'est le cas le plus courant d'une boucle sans fin. Pour sortir de la boucle, il faudra appuyer simultanément sur les touches < Control > < Shift > et < Alternate > . La maîtrise de ce genre de conflit passe par l'instruction EXIT :

```
DO
INPUT nom$
EXIT IF nom$ = "o" OR nom$ = "n"
LOOP
```

La boucle est parcourue une fois, la saisie est enregistrée dans nom\$. Si cette saisie est "o" ou "n", l'instruction EXIT prend son effet et la boucle est interrompue. Dans le cas contraire, c'est parti pour un nouveau tour.

Voici comme prévu l'exercice annoncé. Le programme contient deux boucles DO...LOOP. Modifiez-le de façon à ce que dans la première boucle, seules les valeurs "o" ou "n" soient admises, et que dans la seconde boucle, la saisie d'un âge supérieur à 1000 permette d'arrêter le déroulement du programme :

```
REM un grand bonjour
antique = 100
eternite = 100
INPUT "bonjour, comment t'appelles-tu ?", nom$
PRINT nom$; ", te portes-tu bien ?"
DO
```

```

INPUT reponse$
IF reponse$ = "o"
  PRINT "je m'en réjouis"
ENDIF
IF reponse$ = "n"
  PRINT "j'espère que ça va s'améliorer !"
ENDIF
LOOP
PRINT
PRINT "quel âge as-tu donc ?";
DO
  INPUT age
  IF age > 2 AND age < 20
    PRINT "donc tu sais déjà marcher"
  ENDIF
  pro = age*100/antique
  PRINT "et tu as déjà ";pro;" de ta vie derrière toi"
  PRINT
  beaucoup = eternite-age
LOOP

```

Voici maintenant un exemple de solution :

```

REM un grand bonjour
antique = 100
eternite = 100
INPUT "bonjour, comment t'appelles-tu ?", nom$
PRINT nom$; ", te portes-tu bien ?"
DO
  INPUT reponse$
  IF reponse$ = "o"
    PRINT "je m'en réjouis"
  ENDIF
  IF reponse$ = "n"
    PRINT "j'espère que ça va s'améliorer !"
  ENDIF
  EXIT IF reponse$ = "o" OR reponse$ = "n"
LOOP
PRINT
PRINT "quel âge as-tu donc ?";
DO
  INPUT age
  IF age > 2 AND age < 20
    PRINT "donc tu sais déjà marcher"
  ENDIF
  pro = age*100/antique
  PRINT "et tu as déjà ";pro;" de ta vie derrière toi"
  PRINT
  beaucoup = eternite-age
  PRINT "je te souhaite encore ";beaucoup;" d'années à vivre ";nom$;" !"
  PRINT
  PRINT
  PRINT "comme vous êtes dans un boucle sans fin (DO-LOOP), il faut si vous"

```

```
PRINT "désirez quitter ce prg appuyer simultanément sur control/shift/  
PRINT "alternate ou alors taper que vous avez 1000 ans"  
PRINT  
EXIT IF age > 999  
LOOP
```

## 4.4 Résumé

Vous venez d'apprendre quelques nouveautés ainsi que quelques instructions de base. Nous n'en avons pas encore terminé avec les boucles, mais je vous accorde une petite pause. Voici un survol des procédures de répétition que vous connaissez :

Les boucles avec test au début, dans la mesure où la condition est remplie :

```
WHILE {condition}  
  {instruction(s)}  
WEND  
DO WHILE {condition}  
  {instruction(s)}  
LOOP
```

Les boucles avec test au début, jusqu'à ce que la condition soit remplie :

```
DO UNTIL {condition}  
  {instruction}  
LOOP
```

Les boucles avec test à la fin, jusqu'à ce que la condition soit remplie :

```
REPEAT  
  {instruction(s)}  
UNTIL {condition}
```

```
DO  
  {instruction(s)}  
LOOP UNTIL {condition}
```

Les boucles avec test à la fin, dans la mesure où la condition est remplie :

```
DO  
  {instruction(s)}  
LOOP WHILE {condition}
```

Enfin souvenez-vous des trois touches <Alternate> <Shift> <Control> et de l'instruction EXIT IF.

<pre>DO   {instructions} EXIT IF {conditions} LOOP</pre>
--





## Chapitre 5

### Les répétitions

**V**ous connaissez bien cette situation : vous voulez dire quelque chose à quelqu'un, mais il/elle ne vous écoute pas. Cependant vous voulez absolument le lui dire, si bien que vous répétez votre phrase. Pour insister, vous prolongez votre phrase d'une formule du genre : *combien de fois t'ai-je déjà dit que...* ou même *je t'ai déjà dit cent fois que...*

Si votre phrase n'est toujours pas perçue, et que vous ne voulez pas renoncer, vous voilà embarqué dans une boucle sans fin. Cette situation se présente plus souvent lors de la programmation que dans la vie de tous les jours. A l'origine, il y a souvent une action non réfléchie.

#### 5.1. Pour savoir compter, il faut apprendre

Jusqu'à présent les répétitions dans un programme étaient liées à une condition. Selon les cas, on ne parcourait pas la boucle, on ne la répétait qu'une fois ou plus si nécessaire. Il y a des situations où le nombre de répétitions est très important.

Prenons un exemple parmi les jeux de hasard : pour le loto, il vous faut une suite de nombres dont la valeur ne vous est tout d'abord pas connue et ne peut être que supposée. Au loto, on ne sait qu'une chose : il vous faut une suite de 6 nombres compris entre 1 et 49.

Si vous voulez que votre ordinateur joue pour vous au loto, il faut qu'il gère une suite de nombres qui ait l'air d'être la combinaison gagnante pour le prochain tirage. Dans le basic GFA version 3.0, vous pouvez mettre en mouvement la roue de la fortune grâce à RANDOMIZE qui est un générateur de nombres

aléatoires. La fonction RANDOMIZE n'existe pas dans la version 2.0. Il faut utiliser à la place la fonction RANDOM. La formule :

loto = RANDOM(49) + 1

engendre un nombre entier compris entre 1 et 49 : RANDOM engendre un nombre supérieur à 0 et inférieur à 49. Si on y ajoute 1 nous obtenons les limites de 1 à 49.

La fonction RANDOMIZE dans la version GFA 3.0 sert à initialiser le générateur des nombres aléatoires. A chaque début de programme le générateur des nombres aléatoires est initialisé.

Dans la version 2.0, il faudra écrire RANDOM(x) qui sert à retourner un nombre entier aléatoire compris entre 0 inclus et X exclu.

Revenons maintenant au programme :

**Version 3.0** RANDOMIZE, RAND, RND

**Version 2.0** RANDOM, RND

```
' ALEAS
PRINT "avec RANDOM"
' RANDOM
FOR c=1 TO 6
  loto = RANDOM(49) + 1
  PRINT "chiffre no ";c;" est : ";loto
NEXT c
PRINT
PRINT "avec RAND"
FOR c=1 TO 6
  loto = RAND(49) + 1
  PRINT "chiffre no ";c;" est : ";loto
NEXT c
PRINT
PRINT "avec RND"
FOR c=1 TO 6
  loto = RND(49)
  loto = loto*(49) ! comme avec RND la
  valeur est <= 1 il faut * par 49
  loto = INT (loto) ! arrondir le chiffre
  (partie entière)
  PRINT "chiffre no ";c;" est : ";loto
NEXT c
```

```
' ALEAS
PRINT "avec RANDOM"
' RANDOM
FOR C=1 TO 6
  loto = RANDOM(49) + 1
  PRINT "chiffre no ";c;" est : ";loto
NEXT C
PRINT
PRINT "RAND n'existe pas dans la version
du GFA basic 2.02"
PRINT
PRINT "avec RND"
FOR C=1 TO 6
  loto = RND(49)
  loto = loto*(49) ! comme avec RND la
  valeur est <= 1 il faut * par 49
  loto = INT (loto) ! arrondir le chiffre
  (partie entière)
  PRINT "chiffre no ";c;" est : ";loto
NEXT C
```

Vous avez deux façons de présenter les résultats :

PRINT LOTO'

affichera par exemple

```
3 40 12 26 12 15
```

alors que

```
PRINT;LOTO
```

donnera :

```
3
40
12
26
12
15
```

Avant que vous ne vous précipitiez jusqu'à votre bureau de validation habituel du loto, nous devrions regarder une fois encore de plus près notre programme surtout du côté des limites de la boucle :

```
FOR compteur = 1 TO 6
{
NEXT compteur
```

Comme vous le constatez, FOR et NEXT marquent le début et la fin de la boucle. *Compteur* est la variable qui avance pas à pas de 1 à 6, qui pour ainsi dire accompagne le compteur. Vous pouvez constater par les deux exemples suivants que vous pourriez le faire compter à l'envers ou lui faire sauter des étapes :

### Version 3.0

```
REM pour la version
du GFA Basic 3.03 comptant à l'envers

PRINT "DOWNT0"
FOR c=6 DOWNT0 1
  loto= RANDOM(49) + 1
  PRINT " chiffre no ";c;" EST :";loto
NEXT C
```

### Version 2.0

```
REM pour la version
du GFA Basic 2.02 comptant à l'envers

PRINT "STEP-1"
FOR c=6 TO 1 STEP -1
  loto= RANDOM (49) + 1
  PRINT " chiffre no ";c;" EST :";loto
```

### Version 3.0

```
REM Loto GFA comptant par deux
RANDOMIZE
FOR compteur = 1 TO 11 STEP 2
  loto = RANDOM(49) + 1
  PRINT loto'
NEXT compteur
END
```

### Version 2.0

```
REM Loto GFA comptant par deux
FOR compteur = 1 TO 11 STEP 2
  loto = RANDOM(49) + 1
  PRINT loto'
NEXT compteur
END
```

L'instruction FOR TO NEXT est un compteur qui incrémente ou décrémente la variable à chaque passage. C'est une boucle de comptage. FOR TO NEXT peut être accompagné de STEP. La variable est augmentée de la valeur qui figure avec STEP.

Pour être complet dans nos explications, précisons que FOR...TO sans rien derrière est équivalent à FOR...TO...STEP 1. Et que FOR...DOWNT0 équivaut à FOR...TO...STEP -1. Essayez de voir par vous-même si on peut utiliser des nombres autres qu'entiers avec STEP.

Il y a cependant un os dans ce programme, car il ne vous garantit pas d'obtenir 6 nombres différents ! Lorsque vous en saurez un peu plus, il ne vous sera pas difficile d'améliorer ce petit programme en conséquence. Mais je ne vous garantis pas pour autant que vous gagnerez au loto !

Il en va d'ailleurs de même pour la version modifiée ci-dessous, dans laquelle FOR...NEXT est remplacé par deux types de boucles que vous connaissez :

### Version 3.0

```
REM Loto GFA While
RANDOMIZE
WHILE compteur < 6
  loto = RANDOM(49) + 1
  PRINT loto'
WEND
END
```

### Version 2.0

```
REM Loto GFA While
WHILE compteur < 6
  loto = RANDOM(49) + 1
  PRINT loto'
UNTIL compteur = 6
END
```

### Version 3.0

```
REM Loto GFA Until
RANDOMIZE
REPEAT
  loto = RANDOM(49) + 1
  PRINT loto'
UNTIL compteur = 6
END
```

### Version 2.0

```
REM Loto GFA Until
REPEAT
  loto = RANDOM(49) + 1
  PRINT loto'
UNTIL compteur = 6
END
```

Avez-vous terminé de taper ces petits programmes et de les faire tourner ? Voyez comme vous êtes puni de les avoir repris sans exercer votre sens critique ! en effet les deux boucles contiennent une grosse erreur : elles remplissent sans faille leur mission, mais elles ne veulent plus s'arrêter !

Vous ne pourrez sortir d'une telle boucle sans fin, qu'en appuyant simultanément <Alternate> <Shift> <Control>.

## 5.2. Monter et descendre

Si vous avez bien regardé, vous avez vu que le *compteur* ne change pas. Vous mettez un terme à ces boucles en ajoutant en bonne place l'indication *compteur = compteur + 1* :

### Version 3.0

```
REM Loto GFA While
RANDOMIZE
WHILE compteur < 6
  compteur = compteur + 1
  loto = RANDOM(49) + 1
  PRINT loto'
WEND
END
```

### Version 2.0

```
REM Loto GFA While
WHILE compteur < 6
  compteur = compteur + 1
  loto = RANDOM(49) + 1
  PRINT loto'
WEND
END
```

### Version 3.0

```
REM Loto GFA Until
RANDOMIZE
REPEAT
  compteur = compteur + 1
  loto = RANDOM(49) + 1
  PRINT loto'
UNTIL compteur = 6
END
```

### Version 2.0

```
REM Loto GFA Until
REPEAT
  compteur = compteur + 1
  loto = RANDOM(49) + 1
  PRINT loto'
UNTIL compteur = 6
END
```

Vous voyez bien qu'il faut d'abord apprendre avant de savoir compter ! Le GFA Basic en fait pourtant un peu plus dans ce domaine. Regardez ce que vous pouvez faire lors du comptage en avant :

### Version 3.0

```
REM Loto GFA While
RANDOMIZE
WHILE compteur < 6
  INC compteur
  loto = RANDOM(49) + 1
  PRINT loto'
WEND
END
```

### Version 2.0

```
REM Loto GFA While
WHILE compteur < 6
  INC compteur
  loto = RANDOM(49) + 1
  PRINT loto'
WEND
END
```

### Version 3.0

```
REM Loto GFA Until
RANDOMIZE
REPEAT
  INC compteur
  loto = RANDOM(49) + 1
  PRINT loto'
UNTIL compteur = 6
END
```

### Version 2.0

```
REM Loto GFA Until
REPEAT
  INC compteur
  loto = RANDOM(49) + 1
  PRINT loto'
UNTIL compteur = 6
END
```

Vous avez trouvé ?

*INC* veut dire *incrémenter* et signifie : augmenter la valeur d'une variable. Vous me direz qu'il doit donc y avoir quelque chose pour la faire diminuer :

*DEC* est l'abréviation de *décrémenter* et compte à rebours pas à pas ; vous n'êtes pas encore au bout de vos surprises :

```
REM Monter et descendre les escaliers
PRINT "combien de marches franchis-tu en une enjambée ?"
INPUT "en montant :",marches
WHILE montant < 99
  PRINT montant,
  ADD montant,marches
WEND
PRINT 99
INPUT "en descendant :",marches
descendant = 100
WHILE descendant > 0
  PRINT descendant
  SUB descendant,marches
WEND
PRINT 0
INPUT "",touche$
END
```

L'indication *INPUT ""*, touche est en quelque sorte un ordre d'attente, qui empêche que l'affichage des nombres soit recouvert à la fin du programme par celui de la fenêtre de dialogue. Si vous ne voulez plus voir la colonne de nombres, appuyez tout simplement sur *<Return>*.

Vous pouvez vous même déterminer l'importance des enjambées, et essayer des *nombres-à-virgule*. Vous constatez qu'il y a équivalence entre les instructions :

```
ADD compteur,marches
```

et

```
compteur = compteur + marches
```

ainsi qu'entre :

```
SUB compteur,marches
```

et

```
compteur = compteur - marches
```

Si vous le désirez, vous disposez de plus des instructions suivantes :

```
MUL compteur,marches
```

qui remplace

```
compteur = compteur * marches
```

et

```
DIV compteur,marches
```

qui remplace

```
compteur = compteur / marches
```

Lorsqu'il s'agit d'une incrémentation par pas d'une unité nous n'utiliserons dorénavant que les commandes *INC* et *DEC*.

Ce petit paragraphe traitera sur les fonctions arithmétiques du GFA Basic. Ceci est valable aussi bien pour la version 2.0 que pour la version 3.0 :

#### **L'addition :**

L'instruction est "ADD". Elle additionne les variables entre elles :

```
x = x + y équivaut à ADD x,y
```

#### **La soustraction :**

L'instruction est "SUB". Elle soustrait les variables entre elles :

```
x = x - y équivaut à SUB x,y
```

#### **La division :**

L'instruction est "DIV". Elle divise les variables entre elles :

```
x = x / y équivaut à DIV x,y
```

#### **La multiplication :**

L'instruction est "MUL". Elle multiplie les variables entre elles :

```
x = x * y équivaut à MUL x,y
```

### **5.3. Des boucles sans fin ?**

Restons-en à notre problème *sans fin*. Je vais utiliser pour cela un exemple tiré de n'importe quel manuel de base du Basic et adapté pour le GFA Basic :

```
repete_toujours :  
PRINT "you are"  
GOTO repete_toujours
```

Voilà ce qu'on appelle un saut sans condition préalable. On trouve quelque chose de comparable avec la structure DO...LOOP que voici dans une nouvelle version de notre programme Loto :



## Version 3.0

```
REM Loto sans fin
RANDOMIZE
DO
  loto = RANDOM(49) + 1
  PRINT loto'
LOOP
END
```

## Version 2.0

```
REM Loto sans fin
DO
  loto = RANDOM(49) + 1
  PRINT loto'
LOOP
END
```

Le comptage d'une variable ne vous servirait à rien ici, car ce n'est qu'en appuyant sur < Alternate > < Shift > < Control > que vous pourrez sortir de ce programme qui sinon vous fournira sans arrêt de nouveaux chiffres pour votre loto. La version 2.0 du GFA Basic dispose déjà de cette forme de DO...LOOP.

Les boucles sans fin sont toutefois engendrées le plus souvent par l'énoncé d'une condition qui ne peut pas être remplie. Vous devriez donc tester immédiatement toutes les conditions que vous intégrez dans vos programmes pour savoir si elles peuvent être remplies.

Mais il arrive même qu'un programmeur expérimenté écrive de telles boucles dans un programme. C'est pourquoi il est vivement conseillé, avant de lancer un programme dont on n'est pas sûr, de le sauvegarder par *SAVE* ou *SAVE,A* dans un fichier à part. De cette façon, si on reste bloqué dans une boucle sans fin, on a toujours au moins une copie du dernier état du texte du programme.

Jouons encore un peu avec le hasard : est-ce que ceci vous plaît ?

## Version 3.0

```
REM Loto total
RANDOMIZE
PRINT
REPEAT
  loto = RANDOM(49) + 1
  PRINT loto'
UNTIL loto = 6
END
```

## Version 2.0

```
REM Loto total
PRINT
REPEAT
  loto = RANDOM(49) + 1
  PRINT loto
UNTIL loto = 6
END
```

Cela peut arriver ou ne pas arriver : la probabilité de voir survenir le chiffre 6 pour "loto" est assez forte et croît en tout cas avec le nombre des répétitions. Mais il n'est pas exclu non plus que cette probabilité (loto=6) ne se produise jamais. Il est donc conseillé ici d'introduire une deuxième condition interrompant le déroulement de cette boucle au bout d'un certain temps.

Tournons-nous vers notre programme *un grand bonjour* : quelqu'un pourrait appuyer constamment sur une touche qui ne soit ni "o" ni "n" après *reponse\$* et entrer constamment un chiffre hors limites pour *âge*. Ceci engendrerait une boucle sans fin qu'il faut éviter :

### Version 3.0

```
REM un grand bonjour
antique = 100
eternite = 100
INPUT "bonjour, comment t'appelles-tu
?";nom$
PRINT nom$;"te portes-tu bien ? (o/n)";
WHILE reponse$ < > "o" AND reponse$
< > "n" AND saisie < 10
    INC saisie
    INPUT reponse$
WEND
IF reponse$ = "o"
    PRINT "je m'en réjouis"
ELSE
    PRINT "j'espère que ça va s'améliorer !"
ENDIF
PRINT
PRINT "quel âge as-tu donc ?";
REPEAT
    INC saisie
    INPUT age
UNTIL age > 0 AND age < 100 OR
saisie = 10
IF age > 2 AND age < 20
    PRINT "tu sais donc déjà marcher"
ELSE IF age = 20 AND saisie < = 5
    PRINT "donc tu es un grand"
ENDIF
IF saisie > 5
    PRINT "plaisantin"
ELSE
    pro = age*100/antique
    PRINT "et tu as déjà";pro; "% de ta vie
derrière toi !"
    PRINT
    beaucoup = eternite-age
    PRINT "je te souhaite encore
";beaucoup;" d'années à vivre ";nom$;" !"
ENDIF
END
```

### Version 2.0

```
REM un grand bonjour
antique = 100
eternite = 100
INPUT "bonjour, comment t'appelles-tu
?";nom$
PRINT nom$;"te portes-tu bien ? (o/n)";
WHILE reponse$ < > "o" AND reponse$
< > "n" AND saisie < 10
    INC saisie
    INPUT reponse$
WEND
IF reponse$ = "o"
    PRINT "je m'en réjouis"
ELSE
    PRINT "j'espère que ça va s'améliorer !"
ENDIF
PRINT
PRINT "quel âge as-tu donc ?";
REPEAT
    INC saisie
    INPUT age
UNTIL age > 0 AND age < 100 OR
saisie = 10
IF age > 2 AND age < 20
    PRINT "donc tu sais déjà marcher"
ELSE
    IF age > = 20 AND saisie < = 5
        PRINT "donc tu es un grand"
    ENDIF
    IF saisie > 5
        PRINT "plaisantin !"
    ELSE
        pro = age*100/antique
        PRINT "et tu as déjà";pro;"% de ta vie
derrière toi !"
        PRINT
        beaucoup = eternite-age
        PRINT "je te souhaite encore
";beaucoup;" d'années à vivre ";nom$;" !"
    ENDIF
END
```

Tout utilisateur de ce programme a dix fois la possibilité de saisir une donnée exploitable par l'ordinateur, après quoi on lui communique une appréciation sur son degré de maturité.

Vous n'êtes pas encore à la fin des possibilités offertes par le GFA Basic en matière de structures de boucles : si pour une raison quelconque vous voulez placer dans la boucle elle-même le test de bonne exécution de la condition à remplir, le GFA Basic vous en offre la possibilité, tant dans la version 2.0 que 3.0. Ceci vous permet de sortir d'une boucle quand vous voulez :

#### Version 3.0

```
REM Loto GFA au choix
RANDOMIZE
PRINT "premier nombre : "
DO
  loto = RANDOM(49) + 1
  PRINT loto
  INPUT " assez ? (o/n)";reponse$
  EXIT IF reponse$ = "o"
  PRINT "nombre suivant : "
LOOP
END
```

#### Version 2.0

```
REM Loto GFA au choix
PRINT "premier nombre : ";
DO
  loto = RANDOM(49) + 1
  PRINT loto
  INPUT "assez ? (o/n)";reponse$
  EXIT IF reponse$ = "o"
  PRINT "nombre suivant ; "
LOOP
END
```

Si vous n'en avez pas assez, vous n'avez qu'à appuyer à chaque fois sur <Return> ou <Enter>. Si vous répondez par "o", la boucle s'interrompt à cet endroit. Il est important que EXIT IF ne soit pas encadré immédiatement par IF...ENDIF.

## 5.4. Résumé

Comme vous le voyez, le GFA Basic ne vous contraint pas à éviter les GOTO, mais offre de nombreuses alternatives. Tout dépend de vous : vous vous en tirerez sans doute avec les GOTO, mais vous arriverez plus certainement au but en les évitant dans vos programmes. Pour ma part, je préfère les disques sans rayure.

Vous connaissez maintenant une foule d'instructions de répétition :

Les boucles avec des conditions à remplir :

```
WHILE {condition}
  {instruction(s)}
WEND
```

```
REPEAT
  {instruction(s)}
UNTIL {condition}
```

```
DO WHILE {condition}
  {instruction(s)}
LOOP
```

```
DO UNTIL {condition}
  {instruction}
LOOP
```

```
DO
  {instruction(s)}
LOOP UNTIL {condition}
```

```
DO
  {instruction(s)}
LOOP WHILE {condition}
```

Et vous vous souvenez que des boucles de forme

```
DO
  {instruction(s)}
LOOP
```

sont des boucles sans fin, de même que des boucles ayant des conditions, qui ne peuvent jamais être remplies. Vous devez avoir remarqué que vous ne pouvez sortir de telles situations qu'à l'aide des trois touches <Alternate> <Shift> <Control>.

Vous connaissez aussi les boucles de comptage :

```
FOR {compteur} = {début} TO {fin} STEP {pas}
  {instructions}
NEXT {compteur}
```

```
FOR {compteur} = {début} TO {fin}
  {instructions}
NEXT {compteur}
```

```
FOR {compteur} = {début} DOWNT0 {fin}
  {instructions}
NEXT {compteur}
```

Vous savez que vous pouvez interrompre une boucle à n'importe quel endroit par EXIT IF. Vous connaissez la possibilité par INC et DEC de compter dans le sens de valeurs croissantes ou décroissantes, et de simplifier les additions, soustractions, multiplications et divisions par ADD, SUB, MUL et DIV.

Tout à fait incidemment, vous avez appris quelque chose sur l'utilisation des ordinateurs dans les jeux de hasard : RANDOMIZE vous permet (en GFA 3.0) de lancer le générateur de nombres aléatoires. RANDOM est la fonction qui vous fournit de façon aléatoire un nombre entier compris entre 0 et une valeur déterminée.



## Chapitre 6

### De la manipulation des données

Imaginez votre réaction si vous appreniez que depuis quelque temps on enregistre tout ce que vous dites et faites. Où vous allez, ce que vous aimez etc. *Et alors ? me répondrez-vous je n'ai rien à cacher !. Cela ne vous dérangerait pas un petit peu de savoir que quelque part, on archive tout sur vous ?*

Voilà ce que l'ordinateur peut faire par excellence : saisir des données et les traiter. Malgré son importance, une discussion sur l'utilisation de données archivées dans un ordinateur quelconque n'a pas sa place ici. Nous ne nous préoccupons que de ce que l'ordinateur fait des données qui l'alimentent, en sachant qu'il n'a aucune idée de l'importance qu'elles peuvent revêtir pour nous.

Il ne faut cependant pas perdre de vue que ces données, insignifiantes pour l'ordinateur, pourraient avoir une grande importance pour nous si elles étaient mal utilisées. Car ceci, aussi, est indifférent pour l'ordinateur, qui ne fait que vérifier si les données à traiter sont bien formatées ou si elles sont du type approprié.

#### 6.1. Variables et constantes

J'y reviens toujours : reprenons notre petit programme de conversation avec lequel je vous ai déjà torturé plusieurs fois. Il y a encore des choses à en dire :

## Version 3.0

```

REM rebonjour
PRINT "bonjour, comment t'appelles-tu ?";
INPUT nom$
PRINT nom$; ", comment vas-tu ?";
REPEAT
  INPUT reponse$
  IF reponse$ = "bien"
    PRINT "je m'en réjouis"
  ELSE IF reponse$ = "mal"
    PRINT "j'en suis bien peiné !"
  ELSE
    PRINT "te portes-tu bien ou mal ?"
  ENDIF
UNTIL reponse$ = "bien" OR reponse$ =
"mal"
END

```

## Version 2.0

```

REM rebonjour
PRINT "bonjour, comment t'appelles-tu?";
INPUT nom$
PRINT nom$; ", comment vas-tu?";
REPEAT
  INPUT reponse$
  IF reponse$ = "bien"
    PRINT "je m'en réjouis"
  ELSE
    IF reponse$ = "mal"
      PRINT "j'en suis peiné !"
    ELSE
      PRINT "te portes-tu bien ou mal ?"
    ENDIF
  ENDIF
UNTIL reponse$ = "bien" or reponse$ =
"mal"
END

```

Il y a deux variables dans cet exemple, *nom\$* et *reponse\$*. Mais ce ne sont pas les seules données que l'ordinateur ait à traiter, voyez par vous même :

```

PRINT "bonjour, comment t'appelles-tu ?";
PRINT ", comment vas-tu ?"
PRINT "je m'en réjouis"
PRINT "j'en suis bien peiné !"
PRINT "te portes-tu bien ou mal ?"

```

Chacune de ces instructions d'affichage PRINT est suivie par une chaîne de caractères encadrée par des guillemets :

```

"bonjour, comment t'appelles-tu ?";
", comment vas-tu ?"
"je m'en réjouis"
"j'en suis bien peiné !"
"te portes-tu bien ou mal ?"

```

Vous avez vous-même déterminé ces données lors de l'écriture du programme et les avez saisies dans le programme. Ce qui signifie qu'elles ne pourront plus être modifiées après l'avoir lancé. Si vous désirez que l'ordinateur affiche d'autres textes, vous devrez modifier en conséquence les endroits voulus dans le programme puis le refaire traduire par l'interpréteur du GFA Basic.

A l'inverse des variables, qui peuvent changer à l'intérieur d'un programme, on parle ici de constantes, car ces données sont fixées pour tout le déroulement



du programme. Il en va de même pour les nombres, pour lesquels je vais encore rappeler un exemple déjà vu :

### Version 3.0

```
REM Loto GFA (1)
RANDOMIZE
FOR compteur = 1 TO 6
  loto = RANDOM(49) + 1
  PRINT loto'
NEXT compteur
END
```

### Version 2.0

```
REM Loto GFA
RANDOM
FOR compteur = 1 TO 6
  loto = RANDOM(49) + 1
  PRINT loto'
NEXT compteur
END
```

Les constantes sont ici les chiffres 1, 6, 49 et encore une fois 1. Variions un peu :

### Version 3.0

```
REM Loto GFA (2)
quantite = 6
maximum = 49
RANDOMIZE
FOR compteur = 1 TO quantite
  loto = RANDOM(maximum) + 1
  PRINT loto'
NEXT compteur
END
```

### Version 2.0

```
REM Loto GFA (2)
quantite = 6
maximum = 49
RANDOM
FOR compteur = 1 TO quantite
  loto = RANDOM (maximum) + 1
  PRINT loto'
NEXT compteur
END
```

Maintenant les valeurs limites *quantite* et *maximum* sont devenues des variables. Des valeurs de ce type apparaissent souvent, surtout dans de grands programmes qui seront plus clairs si elles sont fixées dès le début sous des noms significatifs. Ceci vous permet de savoir n'importe où dans votre programme quelle signification revêt le nombre que vous venez d'utiliser. Cette méthode a un autre avantage : si vous devez modifier une valeur -par exemple ici la quantité de nombres que vous voulez obtenir pour jouer au loto ou leur valeur maximum- vous n'avez qu'une modification à faire qui sera valable dans tout le programme.

Il y a cependant un inconvénient, qui peut apparaître dans les programmes comportant beaucoup de noms de variables : s'il vous arrivait d'employer dans une instruction le mot *quantite* à un endroit quelconque de votre programme, cette variable prendrait une valeur non désirée. Pour vous épargner cette déconvenue, vous devriez marquer d'une façon ou d'une autre le nom des variables dont la valeur doit rester constante durant tout le programme, par exemple en ajoutant toujours à la fin du nom un point (.) ou un tiret bas (<sub>0</sub>) :

```
REM Loto GFA (2)
quantite_ = 6
max_ = 49
FOR compteur = 1 TO quantite_
  loto = RANDOM(max_) + 1
  PRINT loto'
NEXT compteur
END
```

## 6.2. Les types de données

*quantite* ou *maximum*, *nom\$* ou *reponse\$* - quels sont les types de données qui existent ? Fondamentalement, le GFA Basic différencie entre les chiffres et les lettres. Il identifie une variable terminée par "\$" comme une chaîne de caractères appelée *String*. Nous connaissons de plus la marque "!" pour une variable booléenne ne pouvant prendre que deux valeurs (vrai ou faux). Faute de quoi l'interpréteur GFA suppose qu'il s'agit d'un nombre. Cette différenciation est importante, car on peut compter avec des nombres :

```
REM String ou nombres
LET x = 1
LET y = x + x
LET x$ = "1"
LET y$ = x$ + x$
PRINT "y = ";y
PRINT "y$ = ";y$
END
```

Vous vous souvenez encore assez de votre cours de mathématiques pour savoir qu'on ne peut pas faire de calcul avec des chaînes de caractères, les Strings ne sont qu'enchaînés l'un après l'autre par le signe "+" (concaténation).

Voilà la signification de LET : il servait à l'origine à distinguer une instruction (nécessitant le signe "=") d'une équivalence marquée par "=". D'autres langages de programmation utilisent en effet des signes différents pour chacune de ces opérations. Nous n'avons pas eu ce problème en écrivant notre exemple de calcul d'une valeur inverse, car on peut se passer de ce signe. Mais il vous arrivera peut-être de vouloir utiliser un des mots du vocabulaire du GFA Basic comme nom d'une variable : LET vous permet alors d'éviter toute confusion. Essayez :

```
LET print = 99
PRINT print
```

Déjà dans les versions antérieures du Basic, tous les nombres n'étaient pas du même type : on distinguait entre les nombres entiers et les *nombres à virgule*, que l'on nommait très élégamment *nombres à virgule flottante*. Ces nombres constituent le domaine des mathématiques usuelles.

Malheureusement l'ordinateur ne travaille pas selon le système décimal, il doit donc d'abord traduire tous les nombres que nous lui soumettons vers son propre système de calcul binaire. Et avant l'affichage sur l'écran ou l'imprimante, il doit reconvertir les nombres écrits dans son format en nombres décimaux. En transformant ainsi des nombres à virgule flottante, on pouvait voir se produire des erreurs dans les valeurs *arrondies* car ces nombres ne peuvent être transformés avec l'exactitude voulue. Prenons par exemple le nombre engendré par la division de 10 par 3 :

3,333....

Combien de chiffres doit on laisser derrière la virgule ? Même le plus puissant des ordinateurs ne peut aller jusqu'à l'infini, il faut donc arrondir la valeur. L'ordinateur préfère de loin les nombres entiers, qui restent exacts et prennent peu de place dans sa mémoire.

Pour que l'interpréteur GFA puisse distinguer entre les sortes de nombres, il faut avoir recours à d'autres signes distinctifs en plus de "\$" et "!". Examinez la table ci-dessous :

#### Signe Type de données et de nombres

- |   |  |
|---|--|
| ! | Valeur booléenne (TRUE ou FALSE) (= BOOLEAN-1 octet)<br>ne peut reconnaître que les valeurs 0 (FALSE) ou (-1) (TRUE)                                   |
|   | Nombre entier <i>court</i> (= BYTE-1 octet) de 0 à 255 (seulement version 3.0)   |
| & | Nombre entier simple (= WORD Mot=2 octets) de -32768<br>jusque + 32767 (version 3.0 seulement)   |
| % | Nombre entier <i>long</i> (= INTEGER-4 octets) de -2147483648 à<br>+ 2147483647  |
| # | Nombre à virgule flottante (= FLOAT-8 octets) d'environ 10-308<br>à environ 10 + 308 (positif ou négatif, avec une exactitude jusqu'à<br>13 positions) |

Ce qui fait que `x!`, `x`, `x%`, `x#` et `x$` représentent six variables différentes, mais par contre `x` et `x#` désignent la même chose car le GFA Basic, en l'absence d'autre spécification, admet d'emblée les nombres non-entiers.

Le nombre d'octets vous indique la place prise par une variable de chaque type dans la mémoire de travail de votre ordinateur. Si par exemple vous disposez d'un Méga ST, il possède 2.000.000 (deux millions) de places dans ses mémoires, mais une bonne partie est déjà occupée par le programme faisant tourner le GFA Basic et aussi par vos programmes.

Les chaînes de caractères (strings) ont un statut particulier, car leur longueur n'est pas déterminée, vu qu'une chaîne peut se composer de zéro, un, plusieurs ou beaucoup de caractères. La plupart des interpréteurs Basic ne permettent qu'une longueur de 255 caractères. Le GFA Basic par contre accepte des chaînes jusqu'à 32767 caractères ; un écran peut afficher  $25 * 80 = 2000$  caractères.

Voulez-vous connaître les caractères dont dispose votre Atari ? Les vétérans du Basic savent que les caractères possédés par l'ordinateur sont mémorisés sous des numéros de code compris entre 0 et 255. Vous pouvez donc afficher l'ensemble des caractères possédés par votre Atari ST par le programme suivant :

```
REM Ensemble des caractères
FOR z = 0 TO 255
  PRINT CHR$(z)
NEXT z
END
```

FOR et NEXT encadrent la commande PRINT. Ceci fait que la variable "z" va prendre successivement toutes les valeurs entre 0 et 255 et provoquer l'affichage des caractères correspondants (CHR\$) tirés de ce qu'on appelle la *table ASCII des caractères de l'Atari*.

Cela ne semble pas tout, car il y a manifestement des caractères invisibles ou sonores. En tout cas vous savez maintenant comment afficher sur l'écran des caractères inaccessibles au clavier. Nous nous en réservons plus loin.

De la même façon que `caractere$ = CHR$(nombre)` vous permet d'afficher le caractère qui se cache derrière un numéro de code compris entre 0 et 255, vous pouvez savoir inversement par `nombre = ASC(caractere$)` le code décimal ASCII d'un caractère.

Il est nécessaire de temps en temps de transformer des données d'un type en données d'un autre type. Par exemple dans notre programme ci-dessus, vous

avez vu la transformation d'un nombre entier compris entre 0 et 255 en un caractère unique. Pour transformer des chiffres en (chaîne de) caractères et inversement, vous utiliserez respectivement *chaine\$=STR\$(nombre)* et *nombre=VAL(chaine\$)*. Le deuxième cas présuppose que la chaîne se compose de nombres, sinon vous obtiendrez pour *nombre* un 0.

Et maintenant vous devez encore savoir ce qui se passe ci-après

### Version 3.0

### Version 2.0

```
REM Arrondir
INPUT "nombre décimal" : ",nombre
PRINT "arrondi à" : ";INT(nombre)
PRINT "avant la virgule" : ";TRUNC
: (nombre)
PRINT "équivalent à" : ";FIX(nombre)
PRINT "après la virgule" : ";FRAC
: (nombre)
PRINT "arrondi juste à" : ";ROUND
: (nombre)
END
```

```
REM Arrondir
INPUT "nombre décimal" : ",nombre
PRINT "arrondi à" : ";INT(nombre)
PRINT "avant la virgule" : ";TRUNC
: (nombre)
PRINT "équivalent à" : ";FIX(nombre)
PRINT "après la virgule" : ";FRAC
: (nombre)
PRINT "arrondi juste à" : ";
D=N-INT(N)
IF D>0.5
N=INT(N)+1
PRINT N
ELSE
N=INT(N)
PRINT N
ENDIF
END
```

INT vous permet d'arrondir un nombre à un nombre entier, TRUNC et FIX effacent les chiffres placés après la virgule. Ceci n'a pas le même effet que INT, et vous le constatez si vous introduisez des nombres négatifs. FRAC fournit la partie d'un nombre située au-delà de la virgule, et vous obtiendrez des chiffres arrondis à l'entier le plus proche, grâce à ROUND. Vous pourrez même arrondir en déterminant vous-même le nombre de chiffres après la virgule grâce à "ROUND(nombre,positions).

La fonction ROUND n'existe pas dans la version 2.0. Voici comment le remplacer dans la version 2.0 :

```
REM arrondir à l'entier le plus proche
INPUT "nombre décimal" : ",N
D=N-INT(N)
IF D>0.5
N=INT(N)+1
PRINT N
ELSE
N=INT(N)
PRINT N
```

```
ENDIF
END
```

Cet exemple suppose que la valeur 3.5 est arrondie à 3.

### 6.3. Définition et débuts des valeurs

A l'inverse des langages comme C, Pascal ou Modula, les variables en Basic n'ont pas besoin d'être définies avant d'être effectivement utilisées par le programme. Il suffit pour cela d'ajouter à leurs dénominations une marque indiquant leur nature à l'interpréteur ou au compilateur. En Basic, les variables ont une valeur par défaut en l'absence de toute autre précision, de zéro (0) lors d'une utilisation comme nombre et sinon de chaîne vide ("" ) en tant que string.

Ceci rend la programmation en Basic un peu plus facile, car on n'est pas obligé de vérifier si les *ingrédients* sont tous prêts.

Lors de la programmation, on peut faire comme le chirurgien en salle d'opération : il n'a qu'à dire *scalpel* pour le recevoir immédiatement dans la main.

Si vous le désirez, vous pourriez procéder à quelques définitions. En effet, dans la version 3.0, vous pouvez convenir avec

```
DEFBIT segment
DEFBYT segment
DEFWRD segment
DEFINT segment
DEFFLT segment
DEFSTR segment
```

que toutes les variables dont les noms apparaissent sous forme d'initiales dans un segment donné doivent ensuite devenir une valeur booléenne (BIT), entier sur 1 octet (BYT), un entier sur 2 octets (WRD), sur 4 octets (INT), un nombre à virgule flottante (FLT) ou une chaîne de caractères (STR). Ce qui vous permet pour les noms de variables fixes de vous passer des marques désignant leur type (! | & % # \$). Par exemple :

**DEFFLT "x-z"** les noms de variables commençant par x, y ou z sont considérés comme des variables à virgule flottante.

**DEFSTR "s"** les noms de variables commençant par un s sont déclarés comme chaînes de caractères.

Je tenais à vous préciser cette possibilité, mais je vous déconseille d'en faire usage : seul le marquage par les suffixes appropriés (! | & % # \$) caractérise de façon claire et catégorique une variable dans tout le programme.

On ne peut pas s'en tirer sans définition lorsqu'on utilise beaucoup de place dans les mémoires, ce qui est le cas lorsque l'on veut mémoriser des données dans un tableau (anglais : array). Prenons par exemple notre programme loto, qui ne nous satisfaisait pas puisqu'il pouvait fournir plusieurs fois de suite le même nombre.

Pour éviter cela, il faut modifier le programme de telle sorte que les nombres déjà sortis soient comparés avec le nouveau nombre. Si ce dernier est déjà sorti, il faut de nouveau procéder à un tirage. Au fur et à mesure qu'ils sortent, les nombres doivent donc être archivés dans un "tableau-loto" qu'il faut d'abord dimensionner par :

```
quantite_ = 6
DIM loto(quantite_)
```

Le jeu parcourt trois boucles : la boucle extérieure (FOR...NEXT) est parcourue exactement autant de fois que la valeur de *quantite\_*. La boucle suivante (REPEAT...UNTIL) recommence un tirage jusqu'à obtention d'un nouveau nombre. On utilise pour contrôler cela une variable booléenne. La boucle interne sert au processus de confrontation :

### Version 3.0

```
REM Loto GFA avec contrôle
quantite_ = 6
max_ = 49
DIM loto(quantite_)
RANDOMIZE
FOR compteur = 1 TO quantite_
  REPEAT
    ok ! = TRUE
    tirage = RANDOM(max_) + 1
    FOR test = 1 TO compteur
      IF tirage = loto(test)
        ok ! = FALSE
      ENDIF
    NEXT test
  UNTIL ok !
  loto(compteur) = tirage
  PRINT loto(compteur)
NEXT compteur
END
```

### Version 2.0

```
REM Loto GFA avec contrôle
quantite_ = 6
max_ = 49
DIM loto (quantite_)
RANDOM
FOR compteur = 1 TO quantite_
  REPEAT
    ok ! = TRUE
    tirage = RADOM(max_) + 1
    FOR test = 1 TO compteur
      IF tirage = loto(test)
        ok ! = FALSE
      ENDIF
    NEXT test
  UNTIL ok !
  loto(compteur) = tirage
  PRINT loto(compteur)
NEXT compteur
END
```

En tant que débutant, vous aurez sans aucun doute quelque difficulté à comprendre la logique de ce programme. Consolez vous en sachant que mon seul objectif était de vous faire connaître les définitions et l'utilisation de variables mémorisées dans un tableau.

L'instruction DIM détermine les dimensions d'un tableau

Après la définition obtenue par DIM, vous avez à votre disposition un certain nombre de variables. Chaque variable peut être appelée et utilisée par son nom et un numéro placé entre parenthèse (indice). Dans la plupart des versions du Basic, le premier indice du tableau vaut 0 ou 1. Vous déterminez cela vous même en utilisant OPTION BASE 0 ou OPTION BASE 1.

Pour les tableaux, il est souvent recommandé de procéder à une initialisation, et de remplir ensuite le tableau. Par exemple après avoir saisi *DIM jour\$(7)* il est recommandé de ne pas laisser vides ("" ) les jours mais de les nommer par leur nom : on pourrait les énumérer un par un, mais je vous recommande la solution suivante :

```
REM semaine
DIM jour$(7)
OPTION BASE 1
FOR t = 1 TO 7
  READ jour$(t)
  PRINT jour$(t)
NEXT t
END
DATA lundi,mardi,mercredi,jeudi
DATA vendredi,samedi,dimanche
```

Faites bien attention à ce que le nombre des données à lire sous READ soit bien indiqué, faute de quoi l'interpréteur GFA vous renverrait un avis d'erreur.

Si tous les éléments d'un tableau ont la même valeur, vous devez utiliser *ARRAYFILL <nom du tableau()>, <valeur numérique>*. Un tableau de caractères ne peut malheureusement pas être complété de cette façon.

Vous pouvez effacer complètement un tableau de n'importe quel type par *ERASE <nom du tableau()>*. La place ainsi libérée dans la mémoire peut être occupée de nouveau pour d'autres tableaux ou d'autres variables. Pour ramener toutes les valeurs à zéro ou vider un tableau, vous utilisez CLEAR.



## 6.4. Les opérateurs

Nous avons fait un peu de mathématiques lorsque nous voulions obtenir la valeur inverse d'un nombre. Vous connaissez je pense les opérations courantes "+ - \* /". Vous allez faire connaissance en plus avec les opérateurs DIV, MOD, "^" et le dernier venu de la version 3.0 "\" Le mieux est de vous rendre compte par vous même de l'effet de ces opérateurs à l'aide de ce petit programme :

```
REM opérateurs mathématiques
PRINT "entrez deux nombres : "
INPUT "premier nombre : ", nombre1
INPUT "deuxième nombre : ", nombre2
resultat = nombre1 + nombre2
PRINT nombre1; "+" ; nombre2; " = "; resultat
resultat = nombre1 - nombre2
PRINT nombre1; "-" ; nombre2; " = "; resultat
resultat = nombre1 * nombre2
PRINT nombre1; "*" ; nombre2; " = "; resultat
resultat = nombre1 / nombre2
PRINT nombre1; "/" ; nombre2; " = "; resultat
resultat = nombre1 \ nombre2
PRINT nombre1; "\" ; nombre2; " = "; resultat
resultat = nombre1 MOD nombre2
PRINT nombre1; "\" ; nombre2; " reste"; resultat
resultat = nombre1 DIV nombre2
PRINT nombre1; "\" ; nombre2; " = "; resultat
END
```

Pour écrire ce programme, servez-vous de la facilité offerte par l'éditeur GFA : saisissez une seule fois

```
resultat = nombre1 + nombre2
PRINT nombre1; "+" ; nombre2; " = "; resultat
```

Amenez ensuite le curseur au début de ce bloc et posez une marque par BlkSta ou <Shift> <F5>. Amenez le curseur à la fin du bloc et marquez soit par BlkEnd soit par <F5>. Voyez si le bloc est marqué, sinon recommencez le processus. Amenez maintenant le curseur là où vous voulez copier le bloc. Par <F4> ou *block* vous faites apparaître une nouvelle ligne de menu :

copy	move	write	l!ist	start	end	^ Del	hide
------	------	-------	-------	-------	-----	-------	------

Copy vous permet de recopier autant de fois que vous le désirez, *block* et *hide* suppriment le marquage. Si vous avez fait trop de copies, vous pouvez en gommer à l'aide de *block* et ^Delete. Si vous voulez déplacer un bloc, utilisez

*block* et *move*. Toutes les options de ce menu peuvent être appelées par leur initiale, sauf *Delete* qui exige -pour des raisons de sécurité- d'actionner en plus la touche <Control>.

Il vous suffit alors de modifier quelques signes et vous vous épargnez beaucoup de travail d'écriture. Vous procédez au mieux à ces modifications en cliquant Insert ou <F8> dans le menu principal ce qui vous amène à Overwr (surimprimer). Faites ensuite tourner le programme plusieurs fois en essayant des nombres différents.

Vous constatez que les opérateurs "\", DIV et MOD vous fournissent toujours des résultats en nombres entiers. DIV fournit la partie entière du résultat de la division (division entière) et MOD le reste de la division (calcul modulo). Vous remarquez que les opérateurs DIV et "\" sont équivalents. Si vous choisissez pour nombre1 et nombre2 des nombres non-entiers, ces opérateurs les arrondissent en nombres entiers. Le dernier opérateur "^" est propre au Basic : il vous permet d'obtenir "nombre1 à la puissance nombre2".

J'en profite ici pour vous présenter les opérateurs de comparaison =, <, >, < >, < =, > = dont vous connaissez sans doute la signification. Je complète seulement cette énumération par un opérateur propre au GFA Basic (v 3.0) : " = " (deux fois le signe égal) qui vous permet de comparer par approximation deux valeurs. Cet opérateur est surtout utile lors de la comparaison de nombres arrondis, et il ne peut en aucun cas être utilisé pour des chaînes de caractères.

Il me faut encore signaler les opérateurs logiques AND (et), OR (ou) et XOR (soit...soit) ainsi que NOT. Si vous avez déjà utilisé un autre Basic, vous les connaissez, si vous êtes un débutant vous les avez déjà rencontrés dans nos instructions IF et vous les rencontrerez encore tout au long de cet ouvrage.

## 6.5 Résumé

Je vous ai expliqué bien des choses sur la manipulation des données, les types de données, les opérations et les définitions. C'était peut-être un peu rapide pour les débutants.

Le résumé qui suit devrait leur fournir un aperçu global.

Vous savez que la valeur des variables peut être modifiée et que la valeur des constantes est fixe. Les signes suivants déterminent le type des données et sont accrochés à la fin des noms des variables :

!	BIT	Valeur booléenne (1 octet) 0 (FALSE) ou 1 (TRUE)
	BYT	Nombre entier "court" (1 octet) de 0 à 255 (version 3.0)
&	WRD	Nombre entier simple (2 octets) entre +/- 32700 (version 3.0)
%	INT	Nombre entier "long" (4 octets) entre +/- 2 Milliards
#	FLT	Nombre à virgule flottante (8 octets)
\$	STR	Chaîne de caractères (string) entre 0 et env. 32700 signes

Si rien ne figure à la fin du nom d'une variable, la valeur par défaut est le #.

Vous savez que dans la version 3.0 vous pouvez définir de façon plus rapide et plus abrégée le type des variables selon leurs dénominations :

```
DEFBIT segment
DEFBYT segment
DEFWRD segment
DEFINT segment
DEFFLT segment
DEFSTR segment
```

Vous avez vu que votre ST dispose de 256 caractères. Chacun de ces caractères possède un code numérique entre 0 et 255 (code décimal) : Vous utilisez *CHR\$(code)* et *ASC(caractere)* pour savoir quel caractère possède quel code numérique et inversement. Cf annexe C.

La fonction *CHR\$(x)* : retourne le caractère dont le code décimal ASCII est x

La fonction *ASC(x\$)* : retourne le code décimal ASCII du premier caractère de la chaîne x\$

De plus *STR\$(nombre)* et *VAL(chaine)* vous permettent de transformer des nombres en chaînes de caractères et inversement. Vous pouvez arrondir des nombres grâce à *INT(nombre)*, *TRUNC(nombre)*, *FIX(nombre)* ou *ROUND(nombre)*. Et si vous avez besoin de la partie décimale, vous l'obtenez par *FRAC(nombre)*.

Vous savez que vous devez définir un tableau par *DIM nom du tableau(indice)* et que vous déterminez par *OPTION BASE 1 ou 0* si l'indice commence à 1 ou 0. Vous savez qu'on ne tient pas compte de valeurs de départ égales à 0 ou de chaînes vides "".

Vous pouvez aussi initialiser d'autres valeurs de départ grâce à *READ* ou *DATA*. S'il s'agit d'un tableau numérique dans lequel tous les éléments ont la même valeur, vous avez recours à *ARRAYFILL*. *CLEAR* ramène toutes les

variables à zéro ou "" et efface complètement le tableau. Si vous ne voulez effacer que certains tableaux, vous utilisez ERASE.

Vous connaissez les opérateurs standard suivants :

pour les nombres	: + - * / \ DIV MOD
pour les chaînes	: + (concaténation)

Vous pouvez mais ne devez pas forcément faire précéder de LET une instruction commençant par "=". Vous connaissez les opérateurs de comparaison =, <, >, <>, <=, >=, les opérateurs logiques AND et OR ou XOR et la négation par NOT.

Dans l'éditeur GFA vous connaissez les opérations de marquage suivantes :

BlkSta	< Shift > < F5 >	pour marquer le début d'un bloc
BlkEnd	< F5 >	pour marquer la fin d'un bloc

Vous avez accès au menu de traitement des blocs par Block ou < F4 > qui vous offre les options

Move	< M >	déplacer
Copy	< C >	copier
Delete	< Control > < D >	effacer
Hide	< H >	ôter les marques.

## Chapitre 7

### Les procédures

**V**ous connaissez le jeu de Léo ? Vous vous souvenez certainement des constructions les plus folles que vous pouviez élaborer grâce aux briques multicolores. Et lorsque cela ne vous plaisait plus, vous pouviez les démonter facilement et les réutiliser pour une autre construction.

Vous avez sans doute joué avec de la pâte à modeler. Vous pouviez là aussi de tout cœur produire des formes sans aucune limitation, ce qui engendrait les plus folles sculptures. Mais à la fin, tout était si bien malaxé que vous ne pouviez plus guère séparer les pâtes des différentes couleurs.

C'est précisément l'aspect *pâte à modeler* du Basic qui réjouit tant d'amateurs. Ce qui en sort ne peut le plus souvent plus être démêlé. Si par contre on construit un programme dès le début brique par brique, le résultat reste compréhensible. On peut ensuite facilement le transformer et l'améliorer, on peut récupérer certaines parties et même les intervertir. Ce style de construction modulaire s'est imposé dans de nombreux domaines surtout techniques.

### 7.1. Les paragraphes

Dans ce livre, je n'utilise qu'une partie du vocabulaire de la langue française. Les mots s'enchaînent en phrases, lesquelles sont groupées en paragraphes. Un groupe de paragraphes forme un chapitre et plusieurs chapitres peuvent aussi former un tout. Naturellement, j'aurais pu aussi écrire tout le texte d'un seul jet, car tout se tient n'est-ce pas ? et j'économiserais ainsi de la place.

Vous devriez à l'avenir faire la même chose pour l'écriture de vos programmes, en divisant l'ensemble du programme en sous-programmes, lesquels seraient divisés en paragraphes etc. Reprenons par exemple notre programme *Bonjour* :

```
REM Bonjour (1)
INPUT "bonjour, comment t'appelles-tu ?";nom$
PRINT nom$;" ,te portes-tu bien ?(o/n)";
REPEAT
  INPUT reponse$
UNTIL reponse$ = "o" OR reponse$ = "n"
IF reponse$ = "o"
  PRINT "je m'en réjouis"
ELSE
  PRINT "j'en suis bien peiné !"
ENDIF
END
```

Nous pouvons subdiviser ce programme de la façon suivante :

```
salutation :
INPUT "bonjour, comment t'appelles-tu ?";nom$
,

question :
PRINT nom$;" ,te portes-tu bien ?(o/n)";
REPEAT
  INPUT reponse$
UNTIL reponse$ = "o" OR reponse$ = "n"
,

exploitation :
IF reponse$ = "o"
  PRINT "je m'en réjouis"
ELSE
  PRINT "j'en suis bien peiné !"
ENDIF
```

Ce qui fait qu'il ne resterait que la charpente suivante :

```
REM bonjour (2)
salutation
question
exploitation
END
,
'suivent les paragraphes dans le détail
:
```

J'entends déjà votre question : pourquoi une telle dépense d'énergie alors que cela n'améliore en rien le déroulement du programme ?

Ce à quoi je vous répondrai : *pourquoi croyez-vous que je me sois donné la peine de trouver toutes les têtes de chapitres et de paragraphes de ce livre ?* Un rouleau imprimé et un texte écrit au kilomètre n'auraient-ils pas suffi ?

L'exemple que nous utilisons est d'ailleurs un très petit programme de salutation. Pour élaborer un vrai programme performant de conversation nous aurions besoin d'un programme beaucoup plus complet. Pensez un peu à toutes les réponses possibles à la question "comment vas-tu ?" et à toutes les questions qu'elles amèneraient ! Ce serait si long qu'on pourrait sans doute remplir des centaines de page d'écran en programmation !

Occupons-nous donc de votre état de santé :

```
question :
IF reponse$ = "o"
' enchainement

ELSE
' precisions
ENDIF
END
```

Nous pourrions encore rediviser ce paragraphe :

```
enchainement :
INPUT "tu vas vraiment bien ?(o/n)";reponse$
IF reponse$ = "o"
PRINT "que pourrait-on bien ajouter à cela ?"
ENDIF
precisions :
REPEAT
PRINT "c'est plutôt psychique ou plutôt physique ?";
INPUT etat$
UNTIL etat$ = "physique" OR etat$ = "psychique"
;
```

Ici aussi on pourrait continuer à subdiviser. Pour comparer, essayez de vous représenter ce que vous feriez avec un programme écrit d'un seul jet et comment vous pourriez y faire des additions !

Comme vous l'avez déjà remarqué, le découpage ci-dessus empêche de faire tourner l'ensemble du programme, car il n'y a encore aucune liaison entre le programme principal et ses sous-programmes, sauf si vous avez laissé tomber END. Nous pourrions restaurer cette liaison avec GOTO :

```
REM Bonjour (3)
GOTO salutation
GOTO question
GOTO exploitation
END
'
salutation :
INPUT "bonjour, comment t'appelles-tu ?";nom$
'
question :
PRINT nom$;" ,te portes-tu bien ?(o/n)";
REPEAT
  INPUT reponse$
UNTIL reponse$ = "o" OR reponse$ = "n"
'
exploitation :
IF reponse$ = "o"
  PRINT "je m'en réjouis"
ELSE
  PRINT "j'en suis bien peiné !"
ENDIF
```

Vous avez essayé cette version ? Apparemment tout se déroule correctement. En réalité, seule l'instruction "GOTO salutation" est exécutée. L'ordinateur n'a pas à exécuter les autres ordres de sauts vu que justement il les a sautés !

## 7.2. Nouvelles instructions

Tant qu'à écrire des sauts, pourquoi toujours des GOTO par-ci, GOTO par là ? Cela vous semble pénible ? Pour revenir par GOTO, vous devez poser de nombreuses marques, et c'est pénible. Moi, je m'en passe car le GFA Basic vous offre la possibilité de formuler les paragraphes ci-dessus comme une procédure :

L'instruction **PROCEDURE** : indique le début d'une procédure (sous programme). Dans le GFA Basic, nous n'utilisons pas de numéros de lignes. C'est pour cela que nous emploierons l'instruction **PROCEDURE**, pour repérer les sous-programmes.

```
PROCEDURE salutation
  INPUT "bonjour, comment t'appelles-tu ?";nom$
RETURN
'
PROCEDURE question
  PRINT nom$;" ,te portes-tu bien ?(o/n)";
  REPEAT
```



```

INPUT reponse$
  UNTIL reponse$ = "o" OR reponse$ = "n"
RETURN
,
PROCEDURE exploitation
  IF reponse$ = "o"
    PRINT "je m'en réjouis"
  ELSE
    PRINT "j'en suis bien peiné !"
  ENDIF
RETURN

```

Vous voyez que je commence par le mot **PROCEDURE**, et termine par **RETURN**. Les procédures *salutation*, *question* et *exploitation* peuvent être appelées comme les instructions ordinaires du GFA Basic. Pour ceux qui ont la version 2.0 du GFA Basic, lisez attentivement la suite du texte avant de taper le programme suivant :

```

REM Bonjour (4)
salutation
question
exploitation
END
,
PROCEDURE salutation
  INPUT "bonjour, comment t'appelles-tu ?";nom$
  RETURN
,
PROCEDURE question
  PRINT nom$; ", te portes-tu bien ?(o/n)";
  REPEAT
    INPUT reponse$
  UNTIL reponse$ = "o" OR reponse$ = "n"
  RETURN
,
PROCEDURE exploitation
  IF reponse$ = "o"
    PRINT "je m'en réjouis"
  ELSE
    PRINT "j'en suis bien peiné !"
  ENDIF
RETURN

```

Essayez de voir ce qui se passe lorsqu'on place la partie principale du programme tout à la fin, en utilisant les instructions de manipulation de blocs.

La version 2.0 du GFA Basic n'offre pas la possibilité d'appeler les procédures en entrant simplement leur nom. Celui-ci doit être précédé d'un **GOSUB** (= Goto subroutine) ou d'un arobas (@) placé juste avant le nom. Ceci est utile lorsqu'il faut distinguer optiquement un nom de procédure d'un nom de

variable. Dans la version 3.0, vous pouvez aussi faire usage de cette possibilité. Le début de notre programme précédent serait donc :

```
REM Bonjour (4)
GOSUB salutation
GOSUB question
GOSUB exploitation
END
```

Les définitions des procédures restent par ailleurs identiques.

Expérimentons maintenant une petite procédure qui vous montrera d'autres possibilités du GFA Basic. Ecrivez d'abord ce petit programme :

#### Version 3.0

```
REM glissements
FOR ligne& = 1 TO 20
  colonne& = 3*ligne&
  PRINT AT (colonne&,ligne&);"glissement"
NEXT ligne&
END
```

#### Version 2.0

```
REM glissements
FOR ligne% = 1 TO 20
  colonne% = 3*ligne%
  PRINT AT (colonne%, ligne%);"
glissement"
NEXT ligne%
END
```

Dans la version 2.0 vous n'utilisez pas la terminaison & pour les variables, vous la remplacez par %. Cela prend certes plus de place, mais permet aux programmes de tourner sans problème, ceci étant valable pour tous vos programmes ultérieurs.

Vous constatez qu'en ajoutant AT, on peut déterminer la position à partir de laquelle l'affichage commence sur l'écran. Par contre, il n'existe pas encore de procédure similaire *input\_at* en GFA Basic : pourquoi ne pas l'écrire nous-mêmes ?

Vous allez me demander ce que deviennent les valeurs placées après une instruction PRINT AT. Il faut bien que ces paramètres arrivent dans la nouvelle procédure définie.

Il serait donc possible d'attribuer des paramètres à une procédure ? Voyez par vous-même :

```
PROCEDURE input_at(colonne&,ligne&,texte$)
  PRINT AT (colonne&,ligne&);
  INPUT "",texte$
RETURN
```

Voyons si cette nouvelle instruction fonctionne bien :

### Version 3.0

```
REM Test Input
PRINT AT(30,10);"comment t'appelles-tu ?
input_at(30,13,nom$)
PRINT AT(30,16);"tu es donc:";nom$
END
,
PROCEDURE input_at(colonne&,ligne&,
texte$)
PRINT AT(colonne&,ligne&);
INPUT """,texte$
RETURN
```

### Version 2.0

```
REM Test Input
PRINT AT (30,10);"comment t'appelles-tu ?
@INPUT AT (30,13,nom$)
PRINT AT (30,16);"tu es donc:";nom$
END
,
PROCEDURE input_at(colonne%,ligne%,
texte$)
PRINT AT(colonne%,ligne%)
INPUT"";texte$
RETURN
```

L'ordinateur semble avoir oublié votre nom. Vous ne l'avez d'ailleurs peut être pas bien vu, car la fenêtre de fin de programme du GFA Basic a caché une partie de l'affichage. Vous éliminez ce problème en écrivant *INPUT touche\$* que vous pouvez déclarer comme une procédure :

```
PROCEDURE attendre
INPUT """,touche$
RETURN
```

Si vous faites tourner ce programme, vous allez constater que l'Atari a effectivement oublié votre nom. Vous proposez de changer la variable *texte\$* en *nom\$* pour que la variable ait partout le même nom ? Essayons cette variante :

### Version 3.0

```
REM Dehors et dedans
LET nom$="personne !"
PRINT AT(30,10);"comment t'appelles-tu ?
input_at(30,13,nom$)
PRINT AT(30,17) : "tu es donc:";nom$
INPUT """,touche$
END
,
PROCEDURE input_at(colonne&,ligne&,
texte$)
PRINT AT(colonne&,ligne&);
INPUT """,nom$
PRINT AT(colonne&,ligne& + 1);"tu
t'appelles donc:";nom$
RETURN
```

### Version 2.0

```
REM Dehors et dedans
LET nom$="personne !"
PRINT AT(30,10);"comment t'appelles-tu ?
@INPUT AT(30,13,nom$)
PRINT AT(30,17) : "tu es donc:";nom$
INPUT """,touche$
END
,
PROCEDURE input_at(colonne%,ligne%,
texte$)
PRINT AT(colonne%,ligne%);
INPUT """,nom$
PRINT AT(colonne%,ligne% + 1);"tu
t'appelles donc:";nom$
RETURN
```

## 7.3. Local ou global ?

Ne soyez donc pas vexé ! Je voulais juste vous montrer que *nom\$* n'a pas le même sens selon qu'il est placé à l'intérieur ou à l'extérieur de la procédure. Vous pouvez maintenant raccourcir la procédure *input\_at* d'un paramètre :

### Version 3.0

```
REM Dehors et dedans
PRINT AT(30,10);"comment t'appelles-tu ?"
input_at(30,13)
PRINT AT(30,17) : "tu es donc : ";nom$
INPUT "",touche$
END

PROCEDURE input_at(colonne&,ligne&)
  PRINT AT(colonne&,ligne&);
  INPUT "",nom$
RETURN
```

### Version 2.0

```
REM Dehors et dedans
PRINT AT(30,10);"comment t'appelles-tu ?"
@input_at(30,13)
PRINT AT(30,17) : "tu es donc : ";nom$
INPUT "",touche$
END

PROCEDURE input_at(colonne%,ligne%)
  PRINT AT(colonne%,ligne%);
  INPUT "",nom$
RETURN
```

Essayons maintenant de comprendre la modification de notre programme : *Input\_at* a seulement deux paramètres *colonne&* et *ligne&*. C'est ce qu'on appelle des variables locales. Elles n'ont un sens qu'au niveau de la procédure *Input\_at*. Leurs contenus ne peuvent pas être transmises au reste du programme. Par contre *nom\$* est une variable du programme et non un paramètre de *Input\_at*. C'est pourquoi sa valeur est identique dans tout le programme.

Si bien qu'on peut certes donner un nom à une procédure et changer ce nom dans la procédure : plus rien n'en filtrera vers le monde extérieur ! Tout sera de nouveau en ordre si on élimine les paramètres non désirés.

Que se passerait-il s'il vous arrivait la même chose dans une autre procédure ? Il n'est pas toujours intéressant de fixer une fois pour toutes un paramètre quelconque. Il devrait être possible de reprendre les paramètres avec des variations, la procédure reprenant une valeur et la transformant en une autre. Voyons cela de plus près :

### Version 3.0 uniquement

```
REM Dehors et dedans
PRINT AT(30,10);"comment t'appelles-tu ?"
input_at(30,13,nom$)
PRINT AT(30,18) : "tu es donc ";nom$
INPUT "",touche$
END
```

```

PROCEDURE input_at(colonne&,ligne&,VAR texte$)
  PRINT AT(colonne&,ligne&);
  INPUT "",texte$
RETURN

```

Le paramètre concerné est maintenant de nouveau dans le programme, mais cette fois il est défini comme variable de programme grâce à VAR. Il porte l'ancienne dénomination *texte\$*. Le mot VAR placé avant *texte\$* indique que *texte\$* est une variable globale. Son contenu doit être transmis au reste du programme. Il peut être nécessaire en effet de transformer le nom d'un paramètre prédéfini en un autre nom. Car le fait de ne pas être limité à un seul nom peut contribuer à rendre une procédure plus facilement, plus universellement utilisable. Cette possibilité n'existe malheureusement que dans la version 3.0. Dans la version 2.0, vous devez essayer de contourner ce problème.

A titre d'exercice, essayez d'utiliser cette procédure *input* de telle sorte que les lettres soient remplaçables par des chiffres. Auriez-vous déjà pensé à rendre possible la saisie dans une même procédure, de chiffres ou de lettres, selon vos besoins ?

Attardons nous encore un peu sur ce *local* ou *global* : le GFA Basic vous offre la possibilité dans les deux versions (2.0 et 3.0) de limiter à certaines procédures le domaine de validité de variables à l'aide de LOCAL :

```

REM global ou local
x = 1
INPUT "entrez un chiffre :",y
@doubler
PRINT "global :";x'y
'
PROCEDURE doubler
  LOCAL x,y
  x = 2
  y = 2*y
  PRINT "local :";x'y
RETURN

```

Pour que vous puissiez ruminer un peu plus, je vous offre encore ces deux variantes, avec des paramètres. Notez que le deuxième exemple fonctionne qu'avec la version 3.0 :

```

REM global ou local (1)
x = 1
INPUT "entrez un chiffre :",y
@doubler(X,Y)
PRINT "dehors :";x'y

```

```
,  
PROCEDURE doubler(x,y)  
  x=2  
  y=2*y  
  PRINT "dedans :";x'y  
RETURN  
  
REM global ou local v.3.0 (2)  
x=1  
INPUT "entrez un chiffre :";y  
doubler(x,y)  
PRINT "dehors :";x'y  
,  
  
PROCEDURE doubler(VARx,y)  
  x=2  
  y=2*y  
  PRINT "dedans :";x'y  
RETURN
```

A quoi bon tout cela me direz-vous ? Pourquoi cette possibilité de définir localement ou globalement ? Une seule définition suffirait bien tout au long d'un programme !

Si vous discutez un jour avec des programmeurs utilisant le langage C, Pascal ou Modula, ils vous diront que c'est un avantage de pouvoir limiter localement la définition et l'utilisation des variables. L'écriture de grands programmes -(ce qui est finalement votre objectif)- fait intervenir une foule de variables, qui devraient toutes porter un nom différent, si elles devaient rester valables durant tout le programme. Le GFA Basic vous permet d'attribuer des noms très longs aux variables (théoriquement 255 caractères).

Souvent cependant un seul nom court suffirait, de quelques caractères, voire même un seul. Pour les variables numériques par exemple, on utilise souvent une lettre "i" ou "n" etc. Il n'est pas à exclure que vous utilisiez, sans vous en rendre compte, le même nom de variable plusieurs fois avec un contenu différent.

Souvent, une variable n'a d'utilité que dans une partie du programme, par exemple dans des boucles numériques. Il est alors souvent peu souhaitable que cette variable ait une *existence* dans l'ensemble du programme et y entraîne le cas échéant des perturbations. Je pense que l'écriture de programmes longs devient très difficile voire impossible si on n'utilise que des variables toujours globalement valables.

Vous apprécierez au fil de votre expérience cette possibilité de limiter le domaine de validité des variables. Le GFA Basic vous l'offre, mais sans vous y contraindre. Elle vous permettra d'employer dans plusieurs procédures une variable "n" ou "x" sans que l'une ait de rapport avec l'autre.

## 7.4. Réutilisation des procédures

Je voudrais vous signaler une deuxième possibilité de positionner le curseur, offerte par le GFA Basic 3.0 :

```
LOCATE ligne,colonne
```

Faites bien attention qu'ici l'ordre est inversé : en premier vient la ligne, en deuxième la colonne ! Notre procédure *input\_at* prend alors la forme suivante :

```
PROCEDURE input_at(colonne&,ligne&,VAR texte$)
  LOCATE ligne&,colonne&
  INPUT "",texte$
RETURN
```

Vérifions maintenant si votre Atari a bien noté cette nouvelle procédure, que ce soit avec PRINT AT ou LOCATE. Il vous sera en effet souvent bien utile d'avoir à votre disposition un certain nombre de procédures prédéfinies pour vos programmes ultérieurs. Marquez pour cela, l'ensemble de la procédure depuis PROCEDURE jusqu'à RETURN dans l'éditeur. Souvenez-vous : on utilise au début BlkSTA ou <Shift> <F5> et à la fin BlkEnd ou <F5>. Le marquage doit être visible. Vous repassez dans le menu de manipulation des blocs par <F4> ou *block*.

En cliquant sur Write ou en appuyant sur <W>, vous voyez s'afficher une fenêtre de dialogue qui vous demande d'attribuer un nom à votre bloc. Ecrivez *INPUT*. Comme dans le cas des options *save*, l'ordinateur se met en devoir d'archiver votre procédure dans un fichier. D'ordinaire, ce fichier recevra l'extension *"LST"*. Vous pouvez la modifier et la distinguer des programmes en écrivant *"PRC"* pour PROCEDURE ce qui vous donnera ici le nom *INPUT.PRC*.

Si vous voulez une copie imprimée de votre bloc, vous cliquez dans Block-Option sur Llist ou <L> : comme dans le menu principal pour l'ensemble du programme, vous obtiendrez une copie papier de votre bloc-procédure. Vérifiez si votre imprimante est bien allumée !

Le bloc ainsi archivé, vous pourrez le rappeler et l'insérer dans le programme à un endroit quelconque par Merge ou <Shift> <F2>. Si vous avez choisi de juxtaposer l'extension *.PRC* au nom de votre procédure, vous devez dans la fenêtre de dialogue, amener votre curseur en cliquant sur la ligne Répertoire : effacer, *\*.LST* en utilisant la touche <Backspace> et remplacer par *\*.PRC*.

Essayez maintenant d'insérer votre procédure *input\_at* à la bonne place dans votre programme. Effacez d'abord la procédure délimitée par des marques à l'aide de Block ou <F4> suivi de Delete ou <Control>-D. Chargez ensuite le bloc grâce à Merge ou <F2> suivi du nom INPUT.PRC : le bloc s'insère alors à l'endroit où est positionné le curseur.

Votre programme devrait ensuite tourner de nouveau de façon tout à fait satisfaisante. Vous supposez que ceci doit être faisable avec n'importe quel autre bloc ? Essayez par vous-même

## 7.5. Résumé

Vous venez d'apprendre à concevoir vos propres procédures, à les archiver dans l'ordinateur et à les réunir dans une bibliothèque.

Vous savez que, pour plus de clarté, vous pouvez expliciter des segments de programme par des lignes de texte précédées d'une apostrophe ou de REM.

Vous pouvez transformer des segments de programme en procédures que vous déterminez en les encadrant par :

```
PROCEDURE Nom
{instructions}
RETURN
```

Ces procédures ne pourront être appelées que par leur nom (3.0), ou en les faisant précéder d'un arobas (@) ou par l'instruction GOSUB. Vous pouvez de plus utiliser cette définition :

```
PROCEDURE Nom (liste de paramètres)
{instructions}
RETURN
```

Vous appelez alors toute cette liste. Mais le classement et le type doivent correspondre aux paramètres prédéfinis. La liste peut contenir aussi bien des paramètres valables uniquement localement que des paramètres de variables (3.0) qui doivent alors être désignés par le préfixe VAR.

Vous pouvez aussi définir des variables avec LOCAL, qui ne seront valables qu'à l'intérieur de la procédure concernée. Toutes les autres variables ont une portée globale.



Vous avez fait connaissance avec deux manipulations dans le menu des blocs :

Write	<W>	pour écrire dans un fichier
List	<L>	pour la sortie sur imprimante

Vous savez aussi que vous pouvez intercaler un bloc dans votre programme en utilisant *Merge*.

Enfin vous savez que, au moyen de *LOCATE ligne,colonne* vous pouvez amener le curseur dans la position voulue sur l'écran pour l'entrée ou la sortie de données. L'instruction *PRINT AT(colonne,ligne)* vous permet de déterminer le début de l'affichage.



## Chapitre 8

### Les fonctions

**D**ans le GFA Basic, les procédures gardent une grande autonomie. Elles *reposent* dans un programme ou un fichier aussi longtemps qu'on ne les appelle pas. Lorsqu'on les met en activité, elles peuvent travailler avec leur propre stock de variables, reprendre des paramètres ou participer aux variables du programme principal. Grâce à cette grande autonomie, on peut ensuite les intégrer dans n'importe quel autre programme.

### 8.1. Pourquoi des fonctions

En définissant des fonctions vous pouvez ajouter des possibilités à tout ce que le GFA Basic vous offre déjà. Vous devriez connaître ce concept depuis vos cours de mathématiques : vous ne vous en souvenez plus ? Par exemple dans :

$f(x)$

"f" était le nom de la fonction et "x" celui de l'argument rattaché à cette fonction. Vous avez déjà entendu cela au sujet des procédures ? Effectivement, les procédures ont une certaine ressemblance avec les fonctions. La grande différence entre les deux, réside dans le fait qu'une fonction vous fournit une valeur et pas une procédure.

Comme le GFA Basic possède une foule de fonctions prédéfinies, je vais vous en citer quelques unes pour éclaircir cette différence. Vous vous souvenez de la fonction INT, qui formait la partie entière d'un nombre quelconque.

```
REM partie entière (1)
INPUT "nombre quelconque: ", nombre
LET entier = INT(nombre)
PRINT "La partie entière est donc: ", entier
END
```

La fonction INT reçoit comme argument la valeur de *nombre* et redonne une valeur sous *entier*.

Une fonction peut par exemple être évaluée. On pourrait renoncer complètement à la variable *entier* :

```
REM entier (2)
INPUT "Nombre quelconque: ", nombre
PRINT "La partie entière est: ", INT(nombre)
END
```

Nous pouvons, si le mot INT ne nous plaît pas, définir notre propre fonction et l'utiliser dans le programme principal :

```
REM entier (3)
INPUT "nombre quelconque: ", nombre
PRINT "la partie entière est: ", FN entier(nombre)
END
'
DEF FN entier(nombre) = INT(nombre)
```

Vous voyez que DEFFN permet de définir et FN d'appeler ce qu'on a défini. Vous pouvez utiliser l'arobas (@) à la place de FN, comme dans les procédures. La ligne a alors l'aspect suivant :

```
PRINT "la partie entière est: ", @entier(nombre)
```

Si vous voulez maintenant faire la même chose avec une procédure, vous allez écrire ceci (il est indifférent qu'une procédure porte le même nom qu'une variable) :

```
REM entier (4)
INPUT "nombre quelconque: ", nombre
@ENTIER(NOMBRE)
PRINT "la partie entière est: ", entier
END
'
PROCEDURE entier(nombre)
    entier = INT(nombre)
RETURN
```

Contrairement à la fonction, on ne peut pas calculer ou évaluer directement une procédure. Prenons un exemple :

```
REM Test de puissance
INPUT "base : ",base
INPUT "exposant : ",expo
PRINT "fonction puissance : ";@puissance(base,expo)
@Puissance(Base,expo)
PRINT "procédure-puissance : ";puissance
END

DEFFN puissance(base,expo) = base ^ expo

PROCEDURE puissance(base,expo)
    puissance = base ^ expo
RETURN
```

Si vous avez bien écouté à l'école, vous savez ce que représentent *base* *expo* et *puissance* pour des valeurs. Ici, nous avons utilisé *puissance* pour trois choses différentes : comme variable globale, comme nom de procédure et comme nom de fonction.

Regardez de plus près la fonction et la procédure afin de bien voir la différence. Considérons les définitions :

```
DEFFN <nom> <valeur>

PROCEDURE <nom>
    {instructions}
RETURN
```

Une telle fonction a quelque chose d'une variable : elle appartient à un type de données; ici, il s'agit d'un nombre décimal, si bien que nous pouvons écrire *DEFFNpuissance#*. On obtient ici une évaluation, ce qui ne serait pas forcément le cas avec la procédure, et même si c'était le cas, on ne peut guère *calculer* une procédure. La façon d'appeler les procédures et les fonctions diffère aussi :

	<nom de la procédure>
ou	@<nom de la procédure>
mais par contre :	
	<nom de variable> = FN <nom de la fonction>
ou	<nom de variable> = @<nom de la fonction>

Une fonction ne peut être utilisée qu'à l'intérieur d'une évaluation ou d'une instruction, comme PRINT par exemple. Contrairement à une instruction, on ne peut l'appeler directement de la façon suivante :

ou	FN < nom de la fonction > @ < nom de la fonction >
----	---

Sous cette dernière forme, l'interpréteur du GFA Basic la prendrait pour une procédure !

## 8.2. Un peu d'acrobatie sur les touches

Regardons quelques fonctions qui appartiennent aux versions standard du GFA Basic. Vous vous souvenez que nous avons déjà compté avec des nombres, et que nous avons même arrangé quelques procédures et quelques fonctions pour le traitement de nombres. Il y a encore d'autres fonctions mathématiques dans le GFA Basic, mais nous allons d'abord nous intéresser à quelques utilitaires.

Permettez-moi de reprendre un exemple un peu défraîchi, mais qui tiendra encore bien la route :

```
REM salut !
INPUT "salut, comment t'appelles-tu ?";nom$
PRINT nom$,"te portes-tu bien ?(o/n)"
REPEAT
  reponse$ = INKEY$
UNTIL reponse$ = "o" OR reponse$ = "n"
IF reponse$ = "o"
  PRINT "je m'en réjouis"
ELSE
  PRINT "j'espère que ça va s'améliorer !"
ENDIF
END
```

Vous venez de découvrir la fonction INKEY\$, et vous avez tout de suite remarqué qu'elle ne possède pas d'argument. Lancez le programme. Vous voulez écrire votre réponse "o" suivie de < Return > mais vous n'en avez pas le temps : sur l'écran s'affiche déjà *j'espère que ça va s'améliorer !* sans que votre "o" ne se soit affiché.

La fonction INKEY\$ a la propriété de lire le premier caractère correct rencontré sur le clavier de l'ordinateur, et de le faire suivre automatiquement de < Return > sans écho sur votre écran, ce qui veut dire que le caractère lu par l'ordinateur ne s'affichera pas. Si vous laissez tourner ce programme, vous pouvez appuyer sur n'importe quelle touche sans que cela soit suivi d'un effet quelconque.

Vous pouvez tout de même interrompre votre programme par <Alternate> <Shift> <Control>.

La boucle ne prend fin qu'avec l'écriture de "o" ou "n". Il suffit d'appuyer sur une seule touche pour obtenir une réponse, ce qui est plus confortable que la saisie avec INPUT. L'inconvénient étant qu'on ne peut entrer qu'un signe et qu'on ne peut plus corriger la saisie. L'instruction INKEY\$ n'attend pas automatiquement une réponse tapée au clavier, elle ne fait que parcourir le clavier, et prend une chaîne vide ("" ) si aucune touche n'est actionnée durant ce court laps de temps :

```
REM INKEY-Test
'
PRINT "1ère touche : ";
clav$=INKEY$
PRINT clav$
'
PRINT "2ème touche : ";
REPEAT
  clav$=INKEY$
UNTIL clav$ < > ""
'
PRINT clav$
END
```

Vous ne voyez pas derrière *1ère touche* la chaîne vide ("" ) retournée par INKEY\$. Ceci correspond au statut *pas de touche*. Si on relie cette fonction à une boucle, le clavier est relu jusqu'à ce qu'on appuie sur une touche (touche < > "" ). En écrivant

```
REPEAT
UNTIL INKEY$ < > ""
```

vous fabriquez une boucle d'attente qui sera relue jusqu'à ce qu'on appuie sur une touche. Si vous préférez travailler avec la souris, vous obtenez le même effet en écrivant :

```
REPEAT
UNTIL MOUSEK
```

La fonction MOUSEK interroge l'état de la souris et, s'il y a eu un clic, demande si une seule ou les deux touche(s) de la souris fu(ren)t actionnée(s). Si oui, la boucle prend fin.

Une autre fonction du GFA Basic vous donnera encore plus de confort dans notre programme *salut* :

```
REM salut !
INPUT "salut, comment t'appelles-tu ?";nom$
PRINT nom$;" ,te portes-tu bien ?(o/n)"
REPEAT
  reponse$ = UPPER$(INKEY$)
UNTIL reponse$ = "O" OR reponse$ = "N"
IF reponse$ = "O"
  PRINT "je m'en réjouis"
ELSE
  PRINT "j'espère que ça va s'améliorer !"
ENDIF
END
```

Ceci vous permet de répondre par "o" ou "n" mais aussi par "O" ou "N". La fonction UPPER\$ transforme toutes les minuscules en majuscules. Cette fonction peut s'appliquer aussi à des chaînes entières de caractères. Vous constaterez qu'elle transforme même les accents : écrivez par exemple :

```
PRINT UPPER$("marché")
```

Pour tester une seule ligne d'instruction, vous pouvez utiliser le mode direct de GFA Basic. Appuyez sur ESC ou cliquez sur "Direct".

Vous entrez alors votre instruction et terminez par <Return>. Puis vous revenez dans l'éditeur par <Esc> <Return>.

Les deux fonctions ci-après vous faciliteront encore votre travail :

```
SPACE$(<Quantité>)
STRING$(<Quantité>,<caractères>)
```

SPACE\$ vous permet de former des chaînes d'un certain nombre d'espaces vides (=spaces), ce qui fonctionne en principe aussi avec STRING\$. L'instruction STRING\$ vous permet de répéter autant de fois que vous le désirez un caractère ou une chaîne de caractères. Pour les caractères simples, vous pouvez vous servir des codes ASCII de l'Atari.

Vous obtiendrez la date et l'heure à l'aide des fonctions DATE\$ et TIME\$, transmises comme des caractères. Dans la version 3.0., ces deux fonctions vous permettent de les mettre à jour si vous voulez.

Vous avez de plus l'instruction SETTIME <heure> <date> : les deux valeurs vous sont communiquées comme des caractères :

l'heure sous la forme "HH:MM:SS"  
(heure minutes secondes)



la date sous la forme "JJ.MM.AA" (jour mois année) ou "JJ.MM.AAAA". Cette instruction permet de fixer la date et l'heure.

Ne soyez pas surpris de constater qu'il s'agit ici de fonctions qui *travaillent* avec des caractères. Car naturellement, les fonctions peuvent être de type string et même reprendre des chaînes comme des paramètres. Vous savez depuis le dernier chapitre que les paramètres des procédures peuvent être des chaînes. Mais vous êtes dans l'erreur si vous en déduisez pour autant qu'un nom de procédure pourrait aussi être entré sous forme de chaîne : le nom d'une procédure n'est qu'un *repère*, une marque. Voyez ci-dessus dans le texte !

### 8.3. Arithmétique des signes

Restons-en aux chaînes de caractères, et essayons de calculer avec elles. En utilisant l'opérateur "+" vous avez vu que les règles sont alors un peu différentes de celles de l'arithmétique ordinaire, et que un plus un ne fera pas forcément deux.

```
REM calcul par signes(1)
INPUT "1er mot";mot.1$
INPUT "2ème mot";mot.2$
somme$ = mot.1$ + mot.2$
PRINT mot.1$;"plus";mot.2$;"=";"somme$
END
```

Faute d'originalité, nous allons prendre une phrase de tous les jours :

```
affirmation$ = "le temps ? vraiment pas beau aujourd'hui"
```

On parle tellement du temps qu'il fait, pourquoi pas ici ? Eventuellement, cette affirmation peut s'avérer totalement fausse, peut-être qu'en

```
realite$ = LEFT$(affirmation$,20) + RIGHT$(affirmation$,16)
```

La *realite\$* résulte de l'*affirmation\$*, puisqu'elle se compose des 20 premiers caractères à partir de la gauche (LEFT\$) et des 16 premiers caractères à partir de la droite (RIGHT\$) - ce qui ressemble à un problème de mots-croisés.

```
Left$ (affirmation$,20)
```

retourne les 20 premiers caractères de *affirmation\$*, c'est-à-dire "le temps ? vraiment".

```
Right$(affirmation$,16)
```

retourne les 16 derniers caractères de affirmation\$, c'est-à-dire "beau aujourd'hui"

Ainsi realite\$ renvoie :

```
"le temps ? vraiment beau aujourd'hui".
```

Vous avez donc vu l'utilité de LEFT\$ et RIGHT\$. Il existe une troisième fonction très utile qui s'appelle MID\$. Elle permet de sélectionner un bout de texte à partir de n'importe quelle position.

Dans notre exemple, créons une variable constat\$

```
constat$ = MID$(affirmation$,1,10) + Mid$(affirmation$,29,16)
```

Le contenu de constat\$ serait : "Le temps ? aujourd'hui".

```
MID$(affirmation$,1,10)
```

retourne 10 caractères à partir du 1er caractère.

```
MID$(affirmation$,29,16)
```

retourne 16 caractères à partir du 29ème caractère de la chaîne affirmation\$.

Si vous n'avez pas bien compris, écrivez donc ces petits programmes et faites les tourner :

```
REM la pluie et le beau temps
affirmation$ = "le temps ? vraiment pas beau aujourd'hui"
PRINT affirmation$
realite$ = LEFT$(affirmation$,20) + RIGHT$(affirmation$,16)
PRINT realite$
constatation$ = MID$(affirmation$,1,10) + MID$(affirmation$,25,16)
PRINT constatation$
MID$(affirmation$,20,4) = "très"
PRINT affirmation$
END
```

## Résumons

```
MID$
```

désigne un segment de la chaîne, délimité par ce qui se trouve entre parenthèses :

```
MID$(< chaîne > , < début > , < longueur > )
```

d'abord le nom de la chaîne, puis le caractère de début puis la longueur du segment. Vous pouvez ainsi parfois remplacer LEFT\$ et RIGHT\$ :

```
realite$ = MID$(affirmation$,1,20) + MID$(affirmation$,25)
PRINT realite$
```

Comme vous le voyez, on peut omettre de préciser la longueur d'une chaîne dans les paramètres, ce qui provoque la saisie implicite de tous les caractères restants. Vous avez sans doute remarqué que MID\$ peut être employé non seulement comme fonction, mais aussi comme instruction :

```
MID$(affirmation$,20,4) = "très"
PRINT affirmation$
```

L'instruction MID\$ provoque la modification de la chaîne de caractères. Vous ne pouvez pas beaucoup vous tromper en entrant les limites avec LEFT\$, RIGHT\$ et MID\$ : si la valeur désignant la longueur du segment dépasse la longueur réelle totale de la chaîne, celle-ci est reprise intégralement. En utilisant MID\$, si le caractère de départ est désigné par une valeur supérieure au total des caractères de la chaîne, vous obtiendrez une chaîne vide affectée de la longueur zéro. Si vous entrez des chiffres à valeur négative (qui aurait cette idée ?) vous recevrez un message d'erreur.

Si vous recherchez un caractère particulier ou un segment dans une chaîne de caractères, utilisez la fonction INSTR.

```
position = INSTR(< chaîne > , < objet recherché > )
```

qui vous indiquera la position de l'objet recherché. Si cet objet est absent de la chaîne, vous verrez s'afficher un zéro. Vous connaîtrez la longueur d'une chaîne de caractères par la fonction

```
longueur = LEN(< chaîne > )
```

Si vous voulez par exemple remplacer un mot par un autre, vous pouvez utiliser ces fonctions comme argument dans les fonctions LEFT\$, MID\$ et RIGHT\$ ainsi qu'avec l'instruction MID\$. Nous voilà maintenant capables de d'améliorer notre programme arithmétique. Après avoir concaténé deux textes, nous allons introduire un troisième mot. Nous supposons bien sûr que ce mot existe bien dans notre texte. D'ailleurs vous pourriez bien essayer tout seul, avant de regarder ce qui suit.

```
REM calcul par signes (2)
INPUT "1er mot";mot.1$
INPUT "2ème mot";mot.2$
somme$ = mot.1$ + mot.2$
PRINT mot.1$;"plus";mot.2$;"=";"somme$
'

INPUT "3ème mot";mot.3$
début = INSTR(somme$,mot.3$)
longueur = LEN(mot.3$)

diff$ = LEFT$(somme$,début-1) + MID$(somme$,début + longueur)
PRINT somme$;"moins";mot.3$;"=";"diff$
END
```

Tout cela sera encore plus élégant si vous utilisez deux fonctions définies par vous-même *add\$* et *sub\$*. Notez que cette élégance n'est valable que dans la version 3.0. Le même programme adapté à la version 2.0 est donné plus bas.

```
REM calcul par signes version 3.0 (3)
INPUT "1er mot";mot.1$
INPUT "2ème mot";mot.2$
somme$ = @add$(mot.1$,mot.2$)
PRINT mot.1$;"plus";mot.2$;"=";"somme$
'

INPUT "3ème mot";mot.3$
diff$ = @sub$(somme$,mot.3$)
PRINT somme$;"moins";mot.3$;"=";"diff$
END
'

DEFFN add$(m1$,m2$) = m1$ + m2$
'

FUNCTION sub$(m1$,m2$)
LOCAL debut,longueur
debut = INSTR(m1$,m2$)
longueur = LEN(m2$)
sub$ = LEFT$(m1$,debut-1) + MID$(m1$,debut + longueur)
RETURN sub$
ENDFUNC
```

Voici le même programme adapté à la version 2.0 :

```
REM calcul par signes version 2.0 (3)
INPUT "1er mot";mot.1$
INPUT "2ème mot";mot.2$
somme$=@add$(mot.1$,mot.2$)
PRINT mot.1$;"+";mot.2$;"=";somme$
,
INPUT "3ème mot";mot.3$
@Sub(somme$,mot.3$)
PRINT somme$;"-";mot.3$;"=";Diff$
END
,
DEFFN add$(m1$,m2$)=1$+2$
,
PROCEDURE sub(m1$,m2$)
LOCAL debut,longueur
debut=INSTR(m1$,m2$)
longueur=LEN(m2$)
diff$=LEFT$(m1$,debut-1)+MID$(m1$,debut+longueur)
RETURN
```

Vous voyez dans la version 3.0 une autre possibilité de définition des fonctions : soit sur une seule ligne (DEFFN) avec attribution directe d'une valeur, ou plus clairement sur plusieurs lignes, encadrées par les mots FUNCTION au début et ENDFUNC à la fin. Vous auriez pu définir la fonction *add\$* en écrivant :

```
FUNCTION add$(m1$,m2$)
  add$=m1$+m2$
  RETURN add$
ENDFUNC
```

Ce type de fonction représente à vrai dire une variante de PROCEDURE : les variables peuvent être définies localement, comme ci-dessus les limites *début* et *longueur* qui ne sont valables qu'à l'intérieur de la fonction *sub\$*. Sinon tout se déroule comme dans une procédure. Il faut seulement ajouter RETURN dans la fonction, suivi d'un paramètre. Voyez ce qui arrive si vous oubliez cette indication.

## 8.4. Résumé

Nous en avons maintenant terminé, vous connaissez la différence entre une procédure et une fonction :

- ❖ les procédures, semblables en cela aux instructions, sont appelées et exécutées directement
- ❖ les fonctions sont attribuées à une variable ou utilisées comme valeurs dans une instruction d'affichage

Vue d'ensemble sur les définitions :

'procédure sur plusieurs lignes :

```
PROCEDURE <nom> (<liste des paramètres>) (version 3.0)
'éventuellement définitions en LOCAL
:
:
RETURN
```

'fonction sur plusieurs lignes :

```
FUNCTION <nom> (<liste des paramètres>)
'éventuellement définitions en LOCAL
:
RETURN <nom>
:
ENDFUNC
```

'fonction sur une ligne :

```
DEFFN <nom> (<liste des paramètres>) = <résultat>
```

Elles sont appelées par :

'procédure :

```
<nom> (<paramètres>) dans la version 3.0
@<nom> (<paramètres>) dans la version 2.0
```

'fonction :

```
<variable> = @<nom> (<paramètres>)
```

Vous savez que les fonctions et procédures peuvent reprendre tous les types de données comme arguments ou paramètres. Les fonctions appartiennent

d'ailleurs elles-mêmes à un certain type de données qu'on peut déterminer soi-même.

Vous utilisez maintenant couramment plusieurs fonctions modifiant des chaînes de caractères.

INKEY\$ transmet, sans écho sur l'écran, le caractère affecté à la touche que vous venez d'actionner. Si vous n'en actionnez aucune, cette fonction génère une chaîne vide (""). Vous pouvez créer une boucle d'attente en écrivant :

```
REPEAT
UNTIL INKEY$ < > ""

REPEAT
UNTIL MOUSEK
```

Pour créer une chaîne de caractères vides, vous utilisez *SPACE\$(quantité)*, et pour créer une chaîne répétant le même caractère *STRING\$( < quantité >, < caractère > )*.

*UPPER\$( < string > )* transforme tous les caractères d'une chaîne en caractères majuscules, même s'ils portent des accents ou un tréma.

*LEFT\$, RIGHT\$* et *MID\$* vous permettent d'obtenir un segment découpé dans une chaîne de caractères depuis la droite, la gauche ou dans la chaîne à partir de l'endroit que vous désirez. *MID\$* vous permet même de modifier des parties d'une chaîne de caractères.

La fonction *INSTR* vous donne la position d'un caractère à l'intérieur d'une chaîne. Si on vous répond *position = 0*, c'est que le caractère est absent. Vous connaîtrez la longueur d'une chaîne grâce à *LEN*. Une chaîne vide aura une longueur = 0.

Vous aurez l'heure et la date par *TIME\$* et *DATE\$* que vous réglerez par *SETTIME < heure >, < date > .*

En tout dernier lieu, rappelez-vous le processus de sauvegarde pour archiver vos blocs de programme, par exemple les fonctions ou les procédures, dans un fichier séparé d'où vous pourrez les recharger pour les insérer dans un programme.

Si vous ne vous en souvenez plus, revoyez de grâce le chapitre précédent !





## Chapitre 9

### Graphisme et son

Aimez-vous lire ? Ou préférez-vous regarder un film ? Les constructions assez abstraites d'un texte parviennent-elles à exciter votre imagination ? Ou vous faut-il plutôt des images ? Que ce soit par la télévision, la vidéo, les magazines illustrés ou les bandes dessinées, l'image l'emporte généralement sur le texte écrit ou parlé. Rappelons de plus l'adage selon lequel un bon plan en dit plus long qu'un millier de mots, ce dont on pourrait douter à la vue de tant de films ou d'images. L'écrit paraît plutôt être démodé car il mobilise plus le cerveau humain qu'une image qui se laisse appréhender de façon concrète.

#### 9.1. Les points et les lignes

Avant de dessiner notre premier point, nous devrions nous donner la peine d'étudier plus attentivement la composition d'un écran. Nous avons appris qu'il se divise normalement en (40 fois 25 =) 1000 ou (80 fois 25 =) 2000 petites cases. Chacune de ces cases se compose d'un certain nombre de points rangés selon une matrice. Selon le type, la surface de l'écran compte :

320 * 200 points en résolution basse (16 couleurs)
640 * 200 points en résolution moyenne (4 couleurs)
640 * 400 points en résolution haute (noir et blanc)

Les exemples graphiques qui vont suivre se rapportent tous à la plus haute résolution, celle de l'écran monochrome.

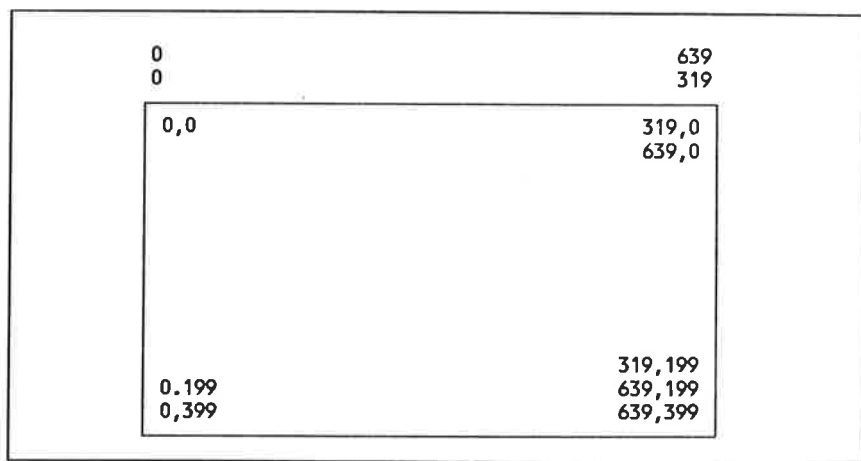
**S'IL VOUS POSSEDEZ UN MONITEUR COULEUR, VOUS ADAPTEREZ LES VALEURS LIMITEES EN DIVISANT PAR DEUX LES COORDONNEES X ET Y.**

Alors que par exemple la commande PRINT utilise ces points pour dessiner un caractère dans le cadre d'une matrice, le mode graphique vous permet d'intervenir point par point.

Pour positionner votre curseur à l'aide de LOCATE, vous écriviez d'abord les coordonnées de la ligne puis celles de la colonne et inversement pour PRINT AT. C'est exactement pareil pour les instructions graphiques : vous devez entrer les coordonnées x et y comme des paramètres.

J'espère que vous vous souvenez tout de même que la position horizontale d'un point est donnée par la valeur x et la verticale par la valeur y. Avant que vous ne commenciez, j'aimerais vous rappeler qu'à la différence de l'affichage du texte où les valeurs commencent à 1, les coordonnées commencent à la valeur 0.

On peut donc diviser l'écran de la façon suivante :



Nous pourrions poser notre premier point en mode direct par l'instruction *PLOT 100,100*, mais ceci serait quelque peu frustrant car on n'y voit quasiment rien.

Il en faut un peu plus ! Voyez un peu l'oeuvre d'art engendrée par le programme suivant, à l'aide de l'instruction RANDOM :

## Version 3.0

```
REM pointillés
RANDOMIZE
FOR i&= 1 TO 10001
  x&=RANDOM(640)
  y&=RANDOM(400)
  PLOT x&,y&
NEXT i&
END
```

## Version 2.0

```
REM pointillés
FOR 1% = 1 TO 10001
  x%=RANDOM(640)
  y%=RANDOM(400)
  PLOT x%,y%
NEXT 1%
END
```

Vous trouvez que c'est trop ? alors faites vous-même vos pointillés ! ou préférez-vous tout de suite dessiner des lignes ? Grâce à :

```
LINE x.début,y.début,x.fin,y.fin
```

vous obtenez une droite selon les coordonnées de début (x debut, y debut) et de fin (x fin, y fin) que vous avez saisies. Ce qui suit vous en fera voir un peu plus :

## Version 3.0

```
REM grille
FOR x&= 10 TO 630 STEP 20
  LINE x&,5,x&,395
NEXT x&
FOR y&= 5 TO 395 STEP 10
  LINE 10,y&,630,y&
NEXT y&
END
```

## Version 2.0

```
REM grille
FOR x%= 10 TO 630 STEP 20
  LINE x%,5,x%,395
NEXT x%
FOR y%= 5 TO 395 STEP 10
  LINE 10,y%,630,y%
NEXT y%
END
```

L'instruction DRAW a le même effet que PLOT et LINE. Vous dessinez un point par *DRAW x,y* et une ligne par *DRAW x.début,y.début TO x.fin,y.fin*.

Si vous voulez relier des droites pour dessiner des polygones, vous écrivez *DRAW TO x.fin,y.fin*. Le début de ligne est fixé par la position du dernier point entré. Vous pouvez après TO accrocher autant de lignes qu'il vous plaira ; ainsi par exemple

```
DRAW 320,10 TO 590,200 TO 590,390 TO 40,390 TO 40,200 TO 320,10
```

vous permet déjà d'obtenir les contours d'une maison, et donc une surface fermée.

## 9.2. Rond ou carré ?

Restons-en au dessin des surfaces : le GFA Basic vous aide là-aussi. Non seulement vous créez des carrés à l'aide de *BOX x.gauche,y.haut,x.droite,y.bas*, mais vous pouvez de plus les remplir soit en les coloriant soit en les grisant par *PBOX* ; vous pouvez arrondir les coins par *RBOX* et combiner les deux par *PRBOX* pour obtenir un carré colorié ou grisé aux coins arrondis :

### Version 3.0

```
REM carrés
FOR i&=0 TO 180 STEP 5
  x.gauche&=i&*640/400
  y.haut&=i&
  x.droit&=640-x.gauche&
  y.bas&=400-y.haut&
  BOX x.gauche&,y.haut&,x.droit&,y.bas&
NEXT i&
END
```

### Version 2.0

```
REM carrés
FOR i%=0 TO 180 STEP 5
  x.gauche%=i%*640/400
  y.haut%=i%
  x.droit%=640-x.gauche%
  y.bas%=400-y.haut%
  BOX x.gauche%,y.haut%,x.droit%,
  y.bas%
NEXT i%
END
```

### Version 3.0

```
REM carrés arrondis
FOR i&=0 TO 200 STEP 2
  x.gauche&=i&*640/400
  y.haut&=i&
  x.droit&=x.gauche&+320
  y.bas&=y.haut&+200
  RBOX x.gauche&,y.haut&,x.droit&,
  y.bas&
NEXT i&
END
```

### Version 2.0

```
REM carrés arrondis
FOR i%=0 TO 200 STEP 2
  x.gauche%=i%*640/400
  y.haut%=i%
  x.droit%=x.gauche%+320
  y.bas%=y.haut%+200
  RBOX x.gauche%,y.haut%,x.droit%,
  y.bas%
NEXT i%
END
```

Si vous voulez faire des ronds encore plus ronds, utilisez l'instruction *CIRCLE x.milieu&,y.milieu&,rayon&* ou *ELLIPSE x.milieu&,y.milieu&, rayon&,y.rayon&*.

Essayez ces deux petits programmes :

### Version 3.0

```
REM arrondis (1)
FOR rayon& = 1 TO 320 STEP 2
  CIRCLE 320,200,rayon&
NEXT rayon&
CLS
FOR rayon& = 320 TO 1 STEP -2
  CIRCLE 320,200,rayon&
NEXT rayon&
END
```

### Version 2.0

```
REM arrondis (2)
FOR rayon% = 1 TO 320 STEP 2
  CIRCLE 320,200,rayon%
NEXT rayon%
CLS
FOR rayon% = 320 TO 1 STEP -2
  CIRCLE 320,200,rayon%
NEXT rayon%
END
```

### Version 3.0

```
REM arrondis (2)
FOR rayon& = 1 TO 320 STEP 2
  ELLIPSE 320,200,rayon&,200
NEXT rayon&
CLS
FOR rayon& = 320 TO 1 STEP -2
  ELLIPSE 320,200,rayon&,200
NEXT rayon&
END
```

### Version 2.0

```
REM arrondis (2)
FOR rayon% = 1 TO 320 STEP 2
  ELLIPSE 320,200,rayon%,200
NEXT rayon%
CLS
FOR rayon% = 320 TO 1 STEP -2
  ELLIPSE 320,200,rayon%,200
NEXT rayon%
END
```

Comme vous le constatez, les valeurs des rayons peuvent être plus grandes que l'écran, une partie du cercle ou de l'ellipse est alors coupée. Ici aussi, les instructions PCIRCLE et PELLIPSE vous permettent de remplir ou griser les surfaces obtenues.

J'ai glissé là-dedans une instruction que je dois maintenant expliquer : CLS (Clear screen) permet de nettoyer l'écran et non la mémoire. Il est recommandé de faire usage de cette instruction de temps en temps.

Le GFA Basic possède aussi une instruction pour dessiner des polygones à 3 angles ou plus : *POLYLINE nombre d'angles, x\_champ(),y\_champ()*. Voyez l'exemple page suivante :

### Version 3.0

```
REM ma maison
DIM x_maison(5),y_maison(5)
FOR i&=0 TO 5
  READ x_maison(i&),y_maison(i&)
NEXT i&
'coordonnées
DATA 320,10
DATA 590,200,590,390
DATA 40,390,40,200
DATA 320,10
POLYLINE 6,x_maison(),y_maison()
'fenêtre et porte
RBOX 80,200,220,300
RBOX 410,200,550,300
RBOX 260,180,370,380
CIRCLE 320,100,40
DEFFILL 1,2,9
FILL 320,170
'attendre la souris
REPEAT
UNTIL MOUSEK
END
```

### Version 2.0

```
REM ma maison
DIM x_maison(5),y_maison(5)
FOR i%=0 TO 5
  READ x_maison(i%),y_maison(i%)
NEXT i%
'coordonnées
DATA 320,10
DATA 590,200,590,390
DATA 40,390,40,200
DATA 320,10
POLYLINE 6,x_maison(),y_maison()
'fenêtre et porte
RBOX 80,200,220,300
RBOX 410,200,550,300
RBOX 260,180,370,380
CIRCLE 320,100,40
DEFFILL 1,2,9
FILL 320,170
'attendre la souris
REPEAT
UNTIL MOUSEK
END
```

"POLYFILL coordonnées,x\_champ(),y\_champ()" engendre directement un polygone colorié ou grisé. Pour *peindre* des surfaces, utilisez FILL x,y, ce qui remplit la surface délimitée par les coordonnées. Le style de remplissage est déterminé par :

DEFFILL <couleur>,<trame de remplissage>,<modèle>.

ceci est aussi applicable aux instructions précédées d'un "P"(= paint) comme par exemple PBOX. Si vous voulez déterminer le style de remplissage, choisissez un nombre entre 0 et 4 :

- |   |   |
|---|---|
| 0 | fond en couleur, pas de trame                 |
| 1 | premier plan en couleur, pas de trame         |
| 2 | pointillé, modèles 1 à 24 de la palette Atari |
| 3 | strié, modèles 1 à 12 de la palette Atari     |
| 4 | modèle à faire soi-même                       |

Vous pouvez aussi déterminer le type des lignes (dans les ronds comme dans les carrés).

Après COLOR <couleur>, vous sélectionnez dans la palette un nombre entre 0 et 1 pour un écran de résolution haute, entre 0 et 3 pour de résolution haute et entre 0 et 15 pour une basse résolution. Vous pouvez affiner vos couleurs grâce à SETCOLOR <registre des couleurs>, <rouge>, <vert>, <bleu>. Vous pouvez déterminer le type de dessin des lignes par :

DECLINE <type de ligne>, <épaisseur>, <forme au début>, <forme à la fin>

par exemple :

0	invisible (s'inscrit dans la teinte du fond)
1	ligne tirée
2	tirets consécutifs
3	pointillés

l'épaisseur peut aller jusqu'à 40 pixels. Les formes au début et à la fin des lignes peuvent être 0 = carré, 1 = flèche et 2 = rond. Pour en savoir plus, procédez à de nombreux essais avec ces instructions.

### 9.3. Oeuvres complètes

Tous ces essais vous ont amené à penser que vous auriez dû embrasser la carrière d'artiste. Je me permets d'attiser la flamme, en vous indiquant ce petit programme, qui vous permettra de ne plus utiliser que la souris pour dessiner :

#### Version 3.0

```
REM dessin avec la souris
DECLINE 1,5,2,2
REPEAT
  MOUSE x%,y%,clic%
  IF clic% = 1
    DRAW x%,y%
  ENDIF
UNTIL clic% = 2
END
```

#### Version 2.0

```
REM dessin avec la souris
DECLINE 1,5,2,2
REPEAT
  MOUSE x%,y%,clic%
  IF clic% = 1
    DRAW x%,y%
  ENDIF
UNTIL clic% = 2
END
```

Pour interrompre ce programme il faudra appuyer sur les touches <control> <shift> <alternate>.

L'instruction MOUSE interroge la position du curseur de la souris et le statut (ici :*click*=") de ses deux touches avec les significations suivantes :

0	pas de touche appuyée
1	touche gauche appuyée
2	touche droite appuyée
3	deux touches appuyées

Vous pouvez dessiner en maintenant enfoncée la touche gauche de la souris (*click* = 1). Le programme prend fin lorsque vous appuyez sur la touche droite (*click* = 2). Je vous laisse le soin de compléter ce petit programme en attribuant d'autres valeurs à DEFLINE ou en utilisant d'autres instructions comme par exemple DEFFILL ou FILL.

Vous pouvez archiver votre oeuvre par SGET écran\$3, ce qui transforme le contenu de l'écran en une seule chaîne (encodée) de 32000 caractères. Si vous visualisez cette chaîne par PRINT, vous ne reconnaîtrez plus rien de votre oeuvre !

Vous pouvez aussi n'archiver que des morceaux de vos créations en utilisant

```
GET x.gauche&,y.haut&,x.droit&,y.bas&,extrait$
```

Il vous faut pour cela indiquer le nom du tableau ainsi que les coordonnées du carré à extraire de l'écran que vous voulez mémoriser. Vous réafficherez ce dessin sur l'écran par l'instruction SPUT écran\$ ou pour les extraits par

```
PUT x.gauche&,y.haut&,extrait$
```

Cela copie une chaîne de 32000 caractères dans la mémoire écran.

Vous n'êtes pas obligé de réafficher cette image à sa place d'origine sur l'écran, il vous suffit de varier ses coordonnées. Cette commande vous permet de plus d'ajouter une option à la suite du nom de la chaîne, option qui déterminera la façon de mixer l'image présente à l'écran avec l'image appelée depuis la mémoire. Quant à ce qui en résultera, essayez par vous-même, votre choix s'étend de 1 à 15.

Si vous combinez l'utilisation de GET et PUT dans un bloc répétitif, vous obtiendrez des dessins animés. Peut-être cet exemple vous plaira-t-il :



## Version 3.0

```

REM en voiture
DEFINE 1,5,2,2
CIRCLE 80,300,50
CIRCLE 300,300,50
CIRCLE 300,300,30
CIRCLE 80,300,30
RBOX 130,220,250,320
RBOX 140,230,240,280
,
REPEAT
UNTIL MOUSEK
,
FOR x% = 0 TO 619 STEP 20
GET x%,210,x% + 350,360,auto$
PUT x% + 20,210,auto$
NEXT x%
END

```

## Version 2.0

```

REM en voiture
DEFINE 1,5,2,2
CIRCLE 80,300,50
CIRCLE 300,300,50
CIRCLE 300,300,30
CIRCLE 80,300,30
RBOX 130,220,250,320
RBOX 140,230,240,280
,
REPEAT
UNTIL MOUSEK
,
FOR x% = 0 TO 619 STEP 20
GET x%,210,x% + 350,360,auto$
PUT x% + 20,210,auto$
NEXT x%
END

```

Si le style de la voiture ne vous plaît pas, modifiez-le vous-même ! Vous avez au moins sur votre écran les deux éléments les plus importants : l'auto et la maison.

Si vous voulez imprimer du texte dans vos dessins, vous pouvez tout d'abord utiliser l'instruction **PRINT**. Texte et curseur seront positionnés grâce à **LOCATE** et **PRINT AT**.

Si le découpage en lignes (de 1 à 25) et en colonnes (de 1 à 80) vous semble trop grossier, servez-vous alors de l'instruction **TEXT x,y, <texte>**, qui vous permettra de placer votre texte au point près.

Vous pouvez de plus faire varier non seulement l'espacement entre les lettres (distance > 0) mais aussi entre les mots (distance < 0) en utilisant :

```
TEXT x,y, <espacement> , <texte>
```

Le type et la grosseur d'écriture sont définis par :

```
DEFTEXT <couleur> , <type> , <positionnement> , <grosseur>
```

Pour définir le type de texte, vous sélectionnez entre 0 et 31 :

0	normal
1	gras (bold)
2	gris (light)
4	italique (italic)

8	souligné
16	évidé (outlined)

Vous pouvez choisir entre quatre positionnements :

0	de gauche à droite (normal)
900	de bas en haut
1800	de droite à gauche (sur la tête)
2700	de haut en bas

Vous avez le choix entre quatre grandeurs des caractères :

4	écriture icône (p. ex. Desktop)
6	petits caractères (taille normale pour les moniteurs couleur)
13	normal (pour les monochromes)
26	grand

Voici une nouvelle modification :

#### Version 3.0

```
REM bonjour
FOR a% = 0 TO 5
  DEFTEXT 1,art,0,13
  TEXT 100*a% + 40,40,"BONJOUR"
  art = 2 ^ a%
NEXT a%

LOCATE 6,6
INPUT "comment t'appelles-tu ?";nom$
DEFTEXT 1,16,0,32
FOR position% = 200 TO 320
  TEXT position%,200,nom$
NEXT position%
WHILE positionnement < 3600
  DEFTEXT 1,16,positionnement,32
  TEXT 320,200,nom$
  ADD positionnement,900
WEND
REPEAT
UNTIL MOUSEK
END
```

#### Version 2.0

```
REM bonjour
FOR a% = 0 TO 5
  DEFTEXT 1,art,0,13
  TEXT 100*a% + 40,40,"BONJOUR"
  art = 2 ^ a%
NEXT a%

PRINT AT(6,6)
INPUT "comment t'appelles-tu ?";nom$
DEFTEXT 1,16,0,32
FOR position% = 200 TO 320
  TEXT position%,200,nom$
NEXT position%
WHILE positionnement < 3600
  DEFTEXT 1,16,positionnement,32
  TEXT 320,200,nom$
  ADD positionnement,900
WEND
REPEAT
UNTIL MOUSEK
END
```

Quand vous en aurez assez de voir votre nom, cliquez tout simplement avec la souris. Vous pourrez d'ailleurs modifier le symbole du curseur de la souris grâce à l'instruction graphique DEFMOUSE, qui engendrera par exemple, 0 une flèche, 2 une abeille, 3 un doigt pointé et 4 une main ouverte.

Vous pouvez même inventer votre propre symbole de curseur grâce à *DEFMOUSE*. Par l'instruction "SPRITE" forme\$. Vous pourrez modifier les objets graphiques indépendants de la souris par l'instruction *SPRITE forme\$*". Mais attention, toutes ces manipulations sont assez compliquées. Si vous voulez apprendre à écrire des chaînes modifiant les symboles, et si vous voulez approfondir cela, il vaudrait mieux utiliser le manuel complet du GFA Basic, et surtout vous entraîner beaucoup auparavant !

## 9.4. L'image ou le son ?

L'instruction "DRAW"(3.0) déplace sur l'écran un crayon imaginaire.

Elle permet de faire encore d'autres choses : avez-vous déjà entendu parler de la tortue graphique ? Un curseur *fouine* sur l'écran à la recherche de certaines instructions. On a comparé cela à une tortue (turtle), qui serait dressée à obéir aux instructions *avance, recule, tourne à gauche, tourne à droite*. Les instructions sont communiquées très sèchement.

Vous pourrez lâcher la bride à votre tortue sur votre écran après avoir entré les instructions et les paramètres adéquats à l'aide de l'instruction DRAW. Les déplacements sont précisés en pixel, les angles à prendre en degrés :

move absolute	ma	x,y	déplacer la tortue jusque x,y absolus
draw absolute	da	x,y	placer le crayon en x,y) et dessiner une ligne de la dernière position à (x,y)
move relative	mr	x,y	se déplacer de x et y pixel par rapport à la dernière position
draw relative	dr	x,y	placer le crayon en (x,y) relatifs (par rapport à la dernière position) et dessiner une ligne de la dernière position à (x,y)
forward	fd	z	avancer de z pixel
backward	bk	z	reculer de z pixel
turn to	tt	w	tourner la tortue jusque l'angle
left turn	lt	w	tourner vers la gauche de w degrés
right turn	rt	w	tourner vers la droite de w degrés
pen down	pd		préparer la tortue à dessiner
pen up	pu		préparer la tortue à se déplacer sans tracer

Etudiez bien ce tableau et essayez le programme suivant :

### Version 3.0 uniquement

```
REM léonard
RBOX 4,4,634,354
SETDRAW 320,200,90
' la tortue est au milieu, orientée vers la droite
REPEAT
  PRINT AT(10,24);"instructions :";SPACE$(55);
  LOCATE 24,24
  LINE INPUT "",tortues
  DRAW tortue$
UNTIL tortue$="ma 0,0"
```

Après avoir lancé le programme, entrez des instructions du tableau donné précédemment.

Entrez par exemple :

pd	<return>
da 100,50	<return>
dr 0,100	<return>
dr 300,0	<return>
da 0,0	<return>
ma 0,0	<return>

SETDRAW permet de placer la tortue, devenue invisible, au milieu de l'écran (320,200). Elle est orientée la tête vers la droite (90). Cette instruction remplace

```
DRAW "ma320,200 tt90"
```

La saisie des abréviations des instructions se fait par LINE INPUT car on utilise des virgules. Le programme se termine lorsque vous avez mis *au panier* (coin en haut à gauche de votre écran) votre tortue par "ma 0,0".

Tout cela vous permet de satisfaire vos yeux mais pas vos oreilles ! si vos dons graphiques vous déçoivent quelque peu, il vous reste la musique, avec l'instruction :

```
SOUND <canal> , <puissance> , <note> , <octave> , <durée>
```

Un exemple sonore :

```
SOUND 1,15,1,4,20
SOUND 2,15,5,4,20
SOUND 3,15,8,4,20
```

L'Atari ST possède trois canaux sonores que vous désignez par leur numéro (de 1 à 3). La puissance sonore s'étale de 1 à 15 - la valeur 0 signifie *silencieusement*. Vous pouvez aussi intervenir sur le volume sonore par le bouton de votre moniteur.

Les notes sont numérotées de 1 à 12. Les musiciens savent ce qu'est une octave, qui se compose de 12 demi-tons :

```
1 = ut, 2 = ut#, 3 = ré, 4 = ré#, 5 = mi, 6 = fa, 7 = fa#, 8 = sol, 9 = sol#, 10 = la,
11 = la#, 12 = si
```

Le son s'étend sur huit octaves (de 1 à 8). Si vous ne connaissez rien à tout cela, faites des essais avec différents nombres. La durée désigne le temps qui s'écoule (en 50ième de seconde) jusqu'à l'exécution de la prochaine instruction. Vous pouvez de nouveau couper le son en écrivant *SOUND <canal>,0,0,0,0*.

La commande WAVE vous permet d'obtenir une suite musicale et d'intervenir sur le timbre. Pour tout cela, vous devriez consulter le manuel complet du GFA Basic et avoir beaucoup de patience. Si vous trouvez que c'est trop compliqué, estimez-vous satisfait avec l'instruction *PRINT CHR\$(7)*.

## 9.5. Résumé

Ce bref chapitre concernant les possibilités graphiques et sonores est loin de vous avoir initié à toutes les subtilités offertes par le GFA Basic. Je ne peux prétendre ici épuiser tout ce qu'on pourrait dire sur ce domaine, qui vous offre un vaste champ de découverte. Mais vous en savez déjà beaucoup :

il existe 3 modes graphiques :

```
320 fois 200 points avec 16 couleurs différentes
640 fois 200 points avec 4 couleurs différentes
640 fois 400 points pour le monochrome
```

Les instructions suivantes vous permettent de déterminer la couleur, le motif de remplissage, la forme des lignes et de l'écriture dans vos images :

COLOR <couleur>
SETCOLOR <registre> , <rouge> , <vert> , <bleu>
DEFFILL <couleur> , <remplissage> , <modèle>
DEFLINE <type de ligne> , <épaisseur> , <début> , <fin>
DEFTXT <couleur> , <type> , <angle> , <grandeur>

Pour dessiner et peindre, vous utilisez :

PLOT x,y	pour des points
LINE x1,y1,x2,y2	pour des lignes
DRAW...TO....	pour des points, des lignes, des périmètres
BOX x1,y1,x2,y2	pour des quadrilatères
RBOX x1,y1,x2,y2	pour des quadrilatères à coins arrondis
POLYLINE n,x(),y()	pour des polygones (trois angles et plus)
CIRCLE x,y,r	pour des cercles
ELLIPSE x,y,r1,r2	pour des ellipses

Des surfaces déterminées sont peintes à l'aide de PBOX, PRBOX, POLYFILL, PCIRCLE et PELLIPSE ; elles peuvent être remplies par FILL.

Vous avez la possibilité d'archiver un dessin ou un morceau de dessin sous forme de chaîne, puis de le rappeler sur votre écran grâce à *SGET\$* ou *SPUT\$* et *GET x1,y1,x2,y2,a\$* ou *PUT x,y,a\$*.

CLS vous permet d'effacer l'écran, et *TEXT x,y, <texte>* d'insérer là où vous le désirez, au point près, une chaîne de caractères. Le paramètre <espacement> vous permet de déterminer l'élongation ou la compression de ce texte écrit.

Vous pouvez manipuler votre souris non seulement par MOUSEK mais aussi par "MOUSE x,y,<statut>". Vous pouvez encore mieux dessiner (dans la version 3.0) grâce à l'instruction DRAW, et même fabriquer des abréviations d'instructions :

ma/mr	x,y	déplacer la tortue
da/dr	x,y	dessiner la tortue
fd/bk	z	z pixel en avant/en arrière
tt/tr	w	tourner la tortue
pd/pu		faire apparaître/disparaître la tortue.

*SETDRAW x,y, <angle>* dans la version 3.0 vous permet de positionner la tortue. Enfin vous accédez à la création sonore par *SOUND <canal> , <puissance sonore> , <note> , <octave> , <durée>*.

## **Partie 3**

### **Le GFA Basic**

**pour les utilisateurs**





## Chapitre 10

### Les structures de programme

**L**es ordinateurs sont avant tout des machines de traitement de données. Car tous les programmes, qu'il s'agisse des programmes d'exploitation ou de traitement, sont dans une certaine mesure des programmes de traitement de données. En effet, tout ce que fait l'ordinateur revient toujours en quelque sorte à traiter des données.

Il n'est donc pas étonnant de constater que beaucoup de programmes ont une construction similaire. Comme vous voilà devenu(e) un(e) technicien(ne) averti(e) sur Atari ST, vous connaissez le concept de *menus déroulants*.

#### 10.1. Un menu

Ce qui correspond le mieux à mon intention de vous donner des bases solides et une grande habileté à confectionner des programmes longs en GFA Basic, c'est précisément l'exemple d'un programme de traitement de données. Il doit être clairement structuré et se composer d'un nombre fini de blocs de programmes.

Beaucoup d'autres programmes sont construits de la même façon : même le traitement de textes ou de graphiques est une forme de traitement des données. Ne renoncez pas à l'insertion d'éléments graphiques ! A l'aide de routines offertes par le GFA Basic, nous allons donner un cadre correct à l'ensemble.

Commençons par planifier notre projet. Soucions-nous du menu principal. Ce dernier doit si possible en un seul coup d'oeil, offrir toutes les (ou la plupart des) options essentielles pour le traitement et la gestion des données et des fichiers.

A titre d'exemple, regardez la barre de menus des applications GEM qui donne accès aux menus déroulants (menus *pull-down*) ou les lignes de menu du GFA Basic.

Vous pouvez alors sélectionner l'option qui vous concerne, soit à l'aide des touches, soit en cliquant sur l'option choisie, avec la souris toujours fournie avec les ordinateurs Atari.

Pour nos exercices, nous nous satisferons tout d'abord d'une fenêtre. Nous disposerons ainsi de toute la surface de l'écran pour écrire notre menu. Nous simplifierons encore notre menu en n'utilisant que la sélection par les touches du clavier.

Pensons tout d'abord aux options nécessaires : que voulons-nous faire avec les données ?

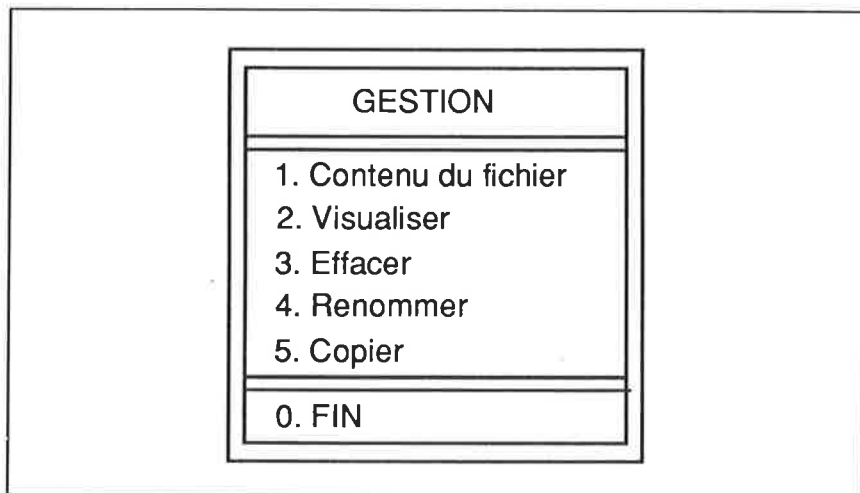
- ❖ En premier lieu, il faut pouvoir sauvegarder les données : nous avons besoin pour cela d'un programme gérant les fichiers. Nous avons besoin d'un programme de saisie pour écrire et entrer les données depuis le clavier dans l'ordinateur. Il nous faut un programme de correction pour pouvoir modifier les données, effacer les données périmées et en ajouter de nouvelles.
- ❖ Il serait intéressant de trier les données, de les ranger selon un ordre déterminé afin d'avoir une meilleure vue d'ensemble et de mieux les retrouver par la suite : il nous faut pour cela un programme de rangement et de recherche des données. Naturellement, nous voulons pouvoir lire les données soit à l'écran soit sur du papier imprimé : il nous faut un programme de sortie-impression.

FICHIER	DONNEES
1. Gestion	6. Enregistrement
2. Sauvegarde	7. tri
3. Chargement	8. Recherche/Remplace
4. Affichage/Impression	
5. aide(s.o.s)	0. Fin
-appuyez sur une touche S.V.P-	

Pour pouvoir traiter et manipuler les données avec souplesse, il nous faut offrir un grand éventail d'options. Les possibilités ci-dessus doivent naturellement être re-subdivisées :

- ⌘ En ce qui concerne la gestion des fichiers, il faudrait pouvoir en obtenir la liste, pouvoir les copier, les effacer et les renommer.
- ⌘ En ce qui concerne les sauvegardes et chargements, il faudrait avoir la possibilité de transférer les fichiers de sauvegarde vers l'ordinateur.
- ⌘ En ce qui concerne la transmission des données, elle devrait pouvoir se faire non seulement vers la sortie parallèle (imprimante), mais aussi vers le port série (modem, autre ordinateur).
- ⌘ En ce qui concerne le traitement, qu'il comprenne la saisie et la sortie des données mais aussi leur correction.
- ⌘ En ce qui concerne le tri : on doit pouvoir ranger, rechercher et remplacer les données selon différents critères.
- ⌘ En cas d'urgence, on dispose de l'option *aide* et enfin de l'option *fin* pour terminer.

Nous avons besoin de sous-menus permettant à l'utilisateur d'exprimer concrètement et facilement ses souhaits. L'un d'eux pourrait prendre la forme suivante :



Vous voyez que quasiment chaque option du menu principal débouche sur un sous-menu, qui ne sera même pas forcément le dernier degré de subdivision.

## 10.2. Options et choix

Je vous préviens tout de suite que nous n'allons pas écrire tous les programmes nécessaires pour toutes les options. N'oubliez pas que mon but est de trouver un exemple vous permettant d'apprendre le GFA Basic ! Vous serez de plus en plus apte à écrire vous-même tout ce qu'il faudra pour adapter ce programme d'exemple à vos besoins - si vous le désirez.

La procédure envisagée pour les menus se divise en deux parties : l'affichage des options disponibles et la sélection de la procédure adéquate.

Il vous faut maintenant traduire cela en GFA Basic. Laissez de côté pour l'instant la présentation graphique des menus dans des cadres, que nous confectionnerons à la fin à l'aide des instructions graphiques GFA.

Et maintenant, au travail ! Fermez votre manuel, et ne l'ouvrez que lorsque vous pensez avoir écrit une procédure valable.

Créons d'abord une procédure de présentation des choix.

```
PROCEDURE options
pos& = 15
PRINT AT(23,5);"FICHER"
PRINT AT(50,8);"DONNEES"
PRINT AT(pos&,8);"1. Gestion"
PRINT AT(43,8);"6. Traitement"
PRINT AT(pos&,10);"2. Sauvegarde"
PRINT AT(43,10);"7. trl"
PRINT AT(pos&,11);"3. Chargement"
PRINT AT(43,11);"8. Recherche/Remplace"
PRINT AT(pos&,13);"4. Affichage/impression"
PRINT AT(pos&,16);"5. Aide (s.O.s.)"
PRINT AT(43,16);"0. Fin"
PRINT AT(pos& + 8,20);" -Appuyez une touche S.V.P.-"
RETURN
```

et maintenant pour pouvoir choisir entre ces options :

```
PROCEDURE choix
INPUT "",choix&
IF choix& = 1
gerer
```

```

ENDIF
IF choix = 2
    sauvegarder
ENDIF
IF choix = 3
    charger
ENDIF
IF choix = 4
    afficher
ENDIF
IF choix = 5
    aider
ENDIF
IF choix = 6
    traiter
ENDIF
IF choix = 7
    trier
ENDIF
IF choix = 8
    rechercher
ENDIF
RETURN

```

Vous pouvez naturellement retenir d'autres noms pour vos procédures si ceux-ci ne vous plaisent pas, puisqu'ils ne sont pas encore prédéfinis. Si vous pensez cela plus pertinent, vous pouvez ne faire qu'une seule procédure en réunissant les deux parties ci-dessus en lui donnant par exemple le nom de *menu principal*.

Je ne vous le conseille pas, car cela ne rendra pas votre programme plus clair, au contraire. A ce stade de l'écriture, notre programme principal ne se compose à vrai dire que des en-têtes *options* et *choix* :

```

REM mini-banque de données
options
choix
END

```

Vous allez certainement sourire ou vous fâcher en voyant le procédé rudimentaire que j'ai utilisé pour écrire le programme de choix, surtout lorsque vous connaîtrez l'instruction ON :

```

PROCEDURE choix
INPUT "",choix&
ON choix GOSUB gerer, sauvegarder, charger, afficher, aider, trier,
rechercher
'tout sur une seule ligne s.v.p. !
RETURN

```

L'un des sous-programmes figurant ci-dessus après GOSUB sera appelé selon le chiffre saisi après *choix\$*. Numéro 1 = gérer, numéro 8 = rechercher. Si vous entrez un zéro ou un chiffre (ici) supérieur à 8. Le programme exécutera l'instruction figurant juste après la ligne de ON, ce qui est très pratique ! Dans sa version 3.0, le GFA Basic vous offre une autre alternative que vous devriez étudier :

### Version 3.0

```
PROCEDURE choix
INPUT "",choix&
SELECT choix&
CASE 1
  gErer
CASE 2
  sauvegarder
CASE 3
  charger
CASE 4
  afficher
CASE 5
  aider
CASE 6
  traiter
CASE 7
  trier
CASE 8
  rechercher
ENDSELECT
RETURN
```

L'instruction SELECT (dans la version 3.0) nous permet de faire des branchements en fonction de la valeur numérique. Endselect nous permet de finir cette condition. Elle doit obligatoirement apparaître.

Les instructions CASE sont multi-conditionnelles. Elles sont parcourues de haut en bas.

## 10.3. De cas en cas

Examinons à la loupe le programme ci-dessus comportant des *embranchements* successifs. On voit tout d'abord au début du bloc, que la variable *choix\$* peut être sélectionnée :

```
SELECT choix$
```

Suivent les différents *cas* parmi lesquels peut s'opérer le *choix*\$. Dans l'instruction CASE il est très important de ne pas écrire sur la même ligne *CASE numéro* et le *nom-de-l'instruction-à-exécuter*.

Remarquez bien qu'un bloc de programme de sélection SELECT..CASE doit se terminer après l'écriture de toutes les options par une marque de fin de sélection. Ici ENDSELECT, tout comme les blocs de IF ou les autres procédures et fonctions.

Pour que vous puissiez maintenant tester votre programme, vous devez relier toutes les procédures ci-dessus par une instruction vide("dummy") :

```
PROCEDURE gerer
RETURN
,
PROCEDURE sauvegarder
RETURN
,
PROCEDURE charger
RETURN
,
PROCEDURE afficher
RETURN
,
PROCEDURE aider
RETURN
,
PROCEDURE traiter
RETURN
,
PROCEDURE trier

RETURN
,
PROCEDURE rechercher
RETURN
```

Vous pourrez par la suite *remplir* chacune de ces *enveloppes*. Du point de vue de la syntaxe, ce programme devrait tourner correctement. Mais quel que soit votre choix, il ne sera pour l'instant suivi d'aucun effet. Les procédures définies ne sont encore que des mirages ! Pour voir quelque chose s'afficher, vous pouvez déjà commencer par intercaler quelques instructions PRINT, mais en tenant compte tout de suite des possibilités du GEM :

```
PROCEDURE gerer
  ALERT 1,"Gestion",1," OK ",back%
RETURN
,
PROCEDURE sauvegarder
```

```

    ALERT 1,"Sauvegarde",1," OK ",back%
RETURN
,
PROCEDURE charger
    ALERT 1,"Chargement",1," OK ",back%
RETURN
,
PROCEDURE afficher
    ALERT 1,"Affichage/Impression",1," OK ",back%
RETURN
,
PROCEDURE aider
    ALERT 3,"Aide (s.o.s.)",1," OK ",back%
RETURN
,
PROCEDURE traiter
    ALERT 2,"Traitement",1," OK ",back%
RETURN
,
PROCEDURE trier
    ALERT 2,"Tri",1," OK ",back%
RETURN
,
PROCEDURE rechercher
    ALERT 2,"Recherche/Remplace",1," OK ",back%
RETURN

```

Tout se passe ensuite comme si on avait réellement la possibilité de choisir. L'instruction ALERT

```
ALERT <symbole> , <texte> , <touche retour> , <touche texte> ,back%
```

permet de gérer ce qu'on appelle des messages d'alarme. Il s'agit en fait d'une fenêtre contenant un texte informatif ainsi qu'un symbole dont le chiffre est choisi entre 0 et 3 :

0	pas de symbole
1	point d'exclamation
2	point d'interrogation
3	panneau STOP

Il y a de plus un ou plusieurs *boutons* (buttons) sur lesquels on peut cliquer avec la souris. Le *bouton return* est généralement dans un cadre souligné de gras, on peut aussi se contenter d'actionner la touche < return > . Il est possible d'écrire un texte devant figurer dans l'un de ces *boutons*. *back%* contient la valeur du bouton sur lequel on clique. Nous n'avons pas encore à introduire cette variable à ce stade du programme, mais il faut déjà écrire *back%* dans la liste des paramètres.



Peut-être n'êtes vous pas tout à fait convaincu de l'utilité de l'instruction SELECT ? Si vous le désirez, vous pouvez en rester à l'instruction ON..GOSUB, ce que vous serez obligé de faire si vous ne disposez que de la version 2.0. Pour moi j'en reste à SELECT..CASE, car cette instruction a l'avantage de permettre des ajouts ultérieurs, comme par exemple :

```
PROCEDURE choix
INPUT "",choix$
SELECT choix$
CASE "1","g"
    gerer
CASE "2","s"
    sauvegarder
CASE "3","c"
    charger
CASE "4","a"
    afficher
CASE "5","O"
    aider
CASE "6","t"
    traiter
CASE "7","l"
    trier
CASE "8","r"
    rechercher
ENDSELECT
RETURN
```

Vous pouvez ainsi sélectionner non seulement en entrant des numéros mais aussi en utilisant les initiales des mots. La procédure sera encore plus confortable si vous écrivez :

```
PROCEDURE choix
REPEAT
    choix$ = UPPER$(INKEY$)
UNTIL choix$ < > ""
SELECT choix$
CASE "1","G"
    gerer
CASE "2","S"
    sauvegarder
CASE "3","C"
    charger
CASE "4","A"
    afficher
CASE "5","O"
    aider.
CASE "6","T"
    traiter
CASE "7","l"
    trier
```

```
CASE "8","R"
  rechercher
ENDSELECT
RETURN
```

De cette façon, tout passe par les touches du clavier, et vous pouvez, en plus des chiffres, utiliser des initiales en minuscules ou en majuscules. Si vous reprenez cette procédure pour votre usage pédagogique, vous devriez attirer l'attention sur la possibilité qu'elle offre d'ajouter d'autres options.

Avec CASE, on peut procéder à des tests sur de plus grandes masses : par exemple CASE "a" TO "z" va parcourir toutes les lettres minuscules de l'alphabet. L'instruction SELECT fonctionne avec des caractères simples mais aussi avec des chaînes pouvant comprendre jusqu'à quatre caractères.

## 10.4. Répétition de la possibilité de choix

Il vous est certainement arrivé lors des tests d'entrer une réponse fausse dans les choix offerts sous CASE, soit par erreur soit délibérément. Il serait alors plus agréable de pouvoir réafficher le menu de sélection.

Vous avez seulement besoin d'une boucle entourant l'instruction de saisie pour empêcher l'entrée de données inexactes. Quels sont les signes et caractères indésirables ?

S'il s'agit de limiter les possibilités de réponse aux touches numériques, l'écriture de la boucle est très simple :

```
REPEAT
  choix$ = INKEY$
UNTIL choix$ >= "0" AND choix$ < "9"
```

Cela paraît par contre bien plus compliqué si l'on veut sélectionner les autres touches, dans l'ensemble des caractères utilisables par l'ordinateur. Celui-ci range en effet les chiffres et lettres dans un certain ordre, mais en les séparant par des signes particuliers. Si vous voulez tenter d'ordonner tous ces caractères et signes pour formuler une condition acceptable, reportez-vous à l'annexe C.

La version 3.0 du GFA Basic vous offre une autre possibilité. Les blocs CASE peuvent être soumis à une condition, si bien qu'on peut adopter la solution suivante : si l'utilisateur appuie sur une des touches retenues dans les blocs

CASE, l'ordinateur exécute la procédure correspondante, sinon il répète la question.

On obtient le programme suivant :

```
PROCEDURE choix
REPEAT
  choix$ = UPPER$(INKEY$)
UNTIL choix$ < > ""
SELECT choix$
CASE "1","G"
  gerer
CASE "2","S"
  sauvegarder
CASE "3","C"
  charger
CASE "4","A"
  afficher
CASE "5","O"
  aider
CASE "6","T"
  traiter
CASE "7","I"
  trier
CASE "8","R"
  rechercher
DEFAULT
  choix
ENDSELECT
RETURN
```

L'instruction Default intervient si aucune des conditions placées entre select...case n'est réalisé ; Il faudra rechoisir jusqu'à ce que la condition soit remplie.

Si aucune des touches adéquates n'a été actionnée (DEFAULT), le menu de choix se réaffiche. Ceci s'appelle une récursion. Je vous mets cependant en garde contre une trop grande fréquence d'erreurs, car l'ordinateur note les réaffichages sur une pile. Il arrive un moment où cette fiche est pleine, et *déborde* ce qui provoque l'affichage d'un message d'erreur : une des mémoires de travail est trop remplie. Dans les cas les plus graves, ceci provoque une interruption du travail en cours, vous devez réinitialiser votre Atari.

Je ne suis pas encore satisfait de ce programme, et vous ? Que se passe-t-il si vous appuyez sur une des bonnes touches provoquant l'exécution de la procédure choisie, qui est toujours pour l'instant réduite à un message d'alarme ? Le programme prend fin !

Et si je devais soit recommencer le même traitement soit en sélectionner un autre ? En effet, après avoir chargé un fichier, je veux pouvoir le traiter, l'afficher si nécessaire et sauvegarder le fichier tel qu'il est après sa mise-à-jour. Pour l'instant, l'utilisateur ne peut pas choisir le moment où il arrête son programme !

Il faudrait disposer d'un *break* ou mieux d'une sortie *en douceur* du programme en cours. Pour pouvoir sortir à volonté du programme par une touche quelconque, il faut que les procédures *options* et *choix* se répètent jusqu'à ce qu'on actionne une touche déterminée. Ceci ressemble fort à une boucle et c'en est une effectivement :

```
REM mini-banque de données
REPEAT
  options
  choix
UNTIL choix$ = "0" OR choix$ = CHR$(27)
END
```

*CHR\$(27)* étend les possibilités de sortie du programme à la touche <Esc>. Cette touche est justement là pour vous permettre de vous échapper (escape). Mais vous pouvez choisir une autre touche pour vous *enfuir*.

Etes-vous satisfait du résultat ? Vous devriez encore tester votre programme. Après avoir obtenu l'affichage des messages d'alarme, essayez de sortir *en douceur*. Vous vous en êtes sorti ? Vous êtes resté coincé ? Alors il faut freiner en appuyant sur les touches :

```
< Control > < Shift > < Alternate >
```

pour sortir de la boucle sans fin. Voilà qui n'est pas bien doux ! Ah ! si nous en étions resté aux chiffres ! Nous sommes maintenant entrés dans une boucle certes élégante mais dont nous ne pouvons sortir sans dégâts !

L'os qui vous reste en travers de la gorge doit se trouver dans la procédure *choix*. Que se passe-t-il si vous n'appuyez sur aucune des touches requises ? Vous faites alors intervenir l'instruction *DEFAULT*, il faut rechoisir et ce jusqu'à ce qu'une des touches adéquates soit actionnée, ce qui provoque l'exécution de la procédure appelée et la fin de la répétition de la grille de choix. Mais le programme principal se poursuit puisque vous n'avez appuyé ni sur "0" ni sur <Esc> (= *CHR\$(27)*) :

```

REPEAT
:
UNTIL choix$ = "0" OR choix$ = CHR$(27)

```

Si bien que vous revenez à la grille de choix. Mais vous aimeriez bien sortir du programme ! Que se passe-t-il si vous appuyez sur "0" ou <Esc> ? Il faut encore et de nouveau choisir ! Car justement, aucune de ces deux touches vous permettant de sortir ne fait partie de celles qui sont retenues comme des réponses correctes. Ce qui fait que la procédure recommence automatiquement, et le programme principal ne prend jamais connaissance de votre action sur les touches de sortie !

Ce processus ne s'arrête plus : nous avons le choix, mais sans pouvoir stopper le programme ! Pour y remédier, il faut faire figurer les deux touches de sortie parmi les touches admises :

```

PROCEDURE choix
  REPEAT
    choix$ = UPPER$(INKEY$)
  UNTIL choix$ < > ""
  SELECT choix$
  CASE "0", 27
    REM vide
  CASE "1", "G"
    gérer
  CASE "2", "S"
    sauvegarder
  CASE "3", "C"
    charger
  CASE "4", "A"
    afficher
  CASE "5", "O"
    aider
  CASE "6", "T"
    traiter
  CASE "7", "I"
    trier
  CASE "8", "R"
    rechercher
  DEFAULT
    choix
  ENDSELECT
RETURN

```

Nous avons ajouté un CASE "0" accompagné du nombre 27 qui désigne la touche <Esc>, après lequel vous ne trouvez plus rien sauf *REM vide*. Lorsque l'interpréteur du GFA Basic ne trouve aucune instruction, il assimile cela à une instruction vide. C'est à dire que l'ordinateur ne fait rien, ce qui est justement l'effet recherché ici. Voilà maintenant que nous avons sorti les touches "0" et

< Esc > de l'instruction DEFAULT, et on peut donc sortir en douceur de ce programme.

## 10.5. Résumé

Ce chapitre vous a surtout permis d'approfondir vos connaissances, mais aussi voir des choses nouvelles. Vous ne connaissez pas seulement la condition simple

```
IF <condition>
  <instructions>
ELSE
  <instructions>
ENDIF
```

mais aussi les conditions multiples

```
ON <valeur> GOSUB <liste des procédures>
```

et aussi dans la version 3.0

```
SELECT <variable>
CASE <valeur>
  <instructions>
  :
DEFAULT
  <instructions>
ENDSELECT
```

Vous savez qu'on peut omettre ELSE et DEFAULT. J'espère que vous n'avez pas oublié que l'instruction SELECT..CASE admet tous les types de données mais seulement des chaînes limitées à quatre caractères, et que CASE < depuis > TO < jusqu'à > vous permet de tester des intervalles de données délimités. Chaque ligne CASE forme un bloc qui peut contenir plusieurs instructions.

Vous savez qu'une procédure du GFA Basic peut s'appeler elle-même et être ainsi récurrente. Vous vous souvenez que CLS (clear screen) sert à effacer l'écran. En tout dernier lieu, avec l'écriture de :

```
ALERT <symbole> , <texte> , <touche return> , <touche texte> ,back%
```

vous avez fait une petite incursion hors du GFA Basic, dans le GEM.

## Chapitre 11

### Les menus, les colonnes et les fenêtres :

#### une bonne optique

Nous n'avons pas encore inauguré notre banque, sans même parler de l'ouvrir ! Pour bien *lancer notre produit*, nous devrions lui donner meilleur aspect. Nous pouvons envisager diverses hypothèses, mais comme notre programme devient de plus en plus long, il nous faut choisir une solution, nous permettant de conserver une certaine clarté dans son déroulement et son écriture.

#### 11.1. Tout doit tenir dans un cadre

Le GFA Basic nous offre une foule d'instructions graphiques ; essayons donc tout de suite de confectionner un cadre. Voici la solution la plus simple :

```
REM mini-banque de données
REPEAT
  options
  BOX 88,42,528,328
  choix
UNTIL choix$ = "0" OR choix$ = CHR$(27)
END
```

Essayons de nous donner un peu plus de mal ! et réécrivons les menus avec tous les enrichissements que nous apportons :

```

PROCEDURE options
  pos& = 15
  PRINT AT(23,5);"FICHIER"
  PRINT AT(50,5);"DONNEES"
  PRINT AT(pos&,8);"1. Gestion"
  PRINT AT(43,8);"6. Traitement"
  PRINT AT(pos&,10);"2. Sauvegarde"
  PRINT AT(43,10);"7. I"
  PRINT AT(pos&,11);"3. Chargement"
  PRINT AT(43,11);"8. Recherche/Remplace"
  PRINT AT(pos&,13);"4. Affichage/Impression"
  PRINT AT(pos&,16);"5. aide (s.O.s.)"
  PRINT AT(43,16);"0. Fin"
  PRINT AT(pos& + 8,20);" -Appuyez une touche S.V.P.-"
RETURN

PROCEDURE choix
REPEAT
  choix$ = UPPER$(INKEY$)
UNTIL choix$ < > ""
CASE "0",27 !Esc
  REM
CASE "1","G"
  gerer
CASE "2","S"
  sauvegarder
CASE "3","C"
  charger
CASE "4","A"
  afficher
CASE "5","O"
  aider
CASE "6","T"
  traiter
CASE "7","I"
  trier
CASE "8","R"
  rechercher
DEFAULT
  choix
ENDSELECT
RETURN

```

Si vous voulez que tout marche bien, vérifiez le nombre exact de caractères vides à l'intérieur du texte à afficher dans la procédure de choix. Si vous ne disposez que de la version 2.0, vous devez encadrer la procédure de *choix* à l'aide de ON..GOSUB ou d'instructions IF.



Tout à fait incidemment, vous avez constaté qu'en plaçant un point d'exclamation (!) on peut ajouter un commentaire à la fin d'une ligne. Nous ferons un large usage de cette possibilité dans le programme principal. Nous allons confectionner tous les cadres nécessaires :

```
REM mini-banque de données avec cadres
,
OPTIONS
,
BOX 88,42,528,328      !cadre principal
BOX 96,50,320,92      !FICHIER
BOX 320,50,520,92     !DONNEES
BOX 96,100,320,220    !Menus 1,2,3,4
BOX 320,100,520,220   !Menus 6,7,8
BOX 96,228,320,268    !AIDE
BOX 320,228,520,268   !FIN
BOX 88,292,528,328    !Touche
,
REPEAT
  choix
UNTIL choix$ = "0" OR choix$ = CHR$(27)
END
```

Ceci devrait constituer notre menu. Vous voulez peut-être encore vous entraîner à faire des cadres ? Si les miens ne vous plaisent pas ou vous paraissent trop larges, n'hésitez pas à les modifier ! RBOX vous permettra d'arrondir les angles, et vous pourrez teinter les surfaces des cadres par FILL.

## 11.2. Encore un menu

Ce menu ne vous convient pas ? Vous êtes tellement gâté par le GEM que vous préféreriez quelque chose comme le GEM Desktop, avec ses menus déroulants et la sélection par la souris ? Il se trouve que là-aussi le GFA Basic peut vous aider. Essayons d'abord de fabriquer un menu du style GEM.

Il nous faut pour cela une chaîne inscrite dans un tableau que nous devons dimensionner : il doit en effet contenir non seulement les textes de vos options mais aussi des chaînes nécessaires au GEM pour l'organisation des menus :

```
PROCEDURE options
,
  DIM texte$(50)
```

Le tableau ainsi délimité va servir à lire (READ) l'ensemble du texte des options figurant dans la liste des données DATA :

```
FOR i=0 TO 24
  READ texte$(i) !porter le texte dans le tableau
NEXT i
'
DATA DESK
DATA Info
DATA -----
DATA 1,2,3,4,5,6,""
DATA FICHER
DATA Gestion
DATA Sauvegarde
DATA Chargement
DATA Affichage/impression
DATA -----
DATA FIN,""
DATA DONNEES
DATA Traitement
DATA Tri
DATA Recherche/Correction
DATA -----
DATA Aide (s.o.s.)
'
MENU texte$()
RETURN
```

L'instruction DATA permet de stocker des valeurs sans grande consommation de place. Ces données sont ensuite lues avec READ. La fonction READ permettra de lire les données contenues dans l'instruction DATA.

Les en-têtes (ici DESK, FICHER, DONNEES) figurent dans la barre du menu principal en haut de l'écran. La chaîne vide ("" ) termine chacun des menus déroulants. Les chiffres contenus dans le menu ouvert par DESK constituent des réserves de place pour des accessoires dont le nombre peut s'élever à six.

La gestion des options est assurée à la fin de la procédure par MENU :

```
MENU texte$() ! reprendre le texte dans le menu
RETURN
```

La procédure de choix est alors modifiée pour s'adapter aux nouvelles conditions. Avant de passer au traitement déclenché par le choix de telle ou telle option, il faut faire revenir à son état antérieur le texte qui a été vidéo-inversé : ceci se fait grâce à MENU OFF :

```
PROCEDURE choix
MENU OFF
```

Ce qui permet de passer au traitement de l'option sélectionnée :

```
choix% = MENU(0)
SELECT choix%
CASE 1
  info
CASE 11
  gerer
CASE 12
  sauvegarder
CASE 13
  charger
CASE 14
  afficher
CASE 19
  traiter
CASE 20
  trier
CASE 21
  rechercher
CASE 23
  aider
ENDSELECT
RETURN
```

La variable *choix%* prend en charge l'indice du point de menu activé par le clic de la souris. Ce numéro est trouvé dans un tableau MENU propre au GEM, qui contient toutes les informations importantes pour la gestion des menus et la manipulation des inscriptions dans les menus. Nous ne nous intéressons qu'à l'inscription portée dans MENU(0), qui est le point sélectionné dans le menu.

Il faut aussi apporter quelques modifications dans le menu principal :

```
REM mini-banque de données en GEM
options
ON MENU GOSUB choix
OPENW 0
```

La fonction OPENW(no) ouvre la fenêtre de numéro (no)

La fonction OPEN O n'ouvre pas une véritable fenêtre, mais déplace simplement l'origine des coordonnées

La fonction CLOSEW ferme la fenêtre de numéro (no)

La fonction CLOSEW O permet de revenir à l'affichage normal sur écran.

Les options s'affichent d'abord. ON MENU GOSUB ajoute un aiguillage à la procédure *choix*, *OPENW 0* replace l'origine des coordonnées directement sous la barre du menu principal, ce qui protège celle-ci contre une écriture en surimpression.

Cela gère une boucle de répétition. Par laquelle grâce à l'instruction ON MENU on demande de surveiller l'intervention des *événements* (ici le clic de la souris sur un des points du menu). Ceci se répète jusqu'à ce que l'on actionne la touche droite de la souris ou que l'on sélectionne le point FIN à l'aide de la touche gauche :

```
REPEAT
  ON MENU
UNTIL MOUSEK = 2 OR choix% = 16
```

### 11.3. Listing et folding

Nous avons maintenant réuni les matériaux essentiels. Vous voulez certainement voir si vous avez bien appris le maniement du GEM. C'est pourquoi je vous donne ici l'ensemble du texte du programme :

```
REM mini-banque de données en GEM
'
options
ON MENU GOSUB choix
OPENW 0
REPEAT
  ON MENU
UNTIL MOUSEK = 2 OR choix% = 16
'touche droite de la souris ou point "fin" du menu
END
'
'
PROCEDURE options
  DIM texte$(50)
  FOR i=0 TO 24
    READ texte$(i)
  NEXT i
'
  DATA DESK
  DATA Info
  DATA -----
  DATA 1,2,3,4,5,6,""
  DATA FICHIER
```

```

DATA Gestion
DATA Sauvegarde
DATA Chargement
DATA Affichage/impression
DATA -----
DATA FIN,""
DATA DONNEES
DATA Traitement
DATA Tri
DATA Recherche/Correction
DATA -----
DATA Aide (s.o.s.)
,
MENU texte$()
RETURN
,
PROCEDURE choix
MENU OFF
choix% = MENU(0)
SELECT choix%
CASE 1
    info
CASE 11
    gerer
CASE 12
    sauvegarder
CASE 13
    charger
CASE 14
    afficher
CASE 19
    traiter
CASE 20
    trier
CASE 21
    rechercher
CASE 23
    aider
ENDSELECT
RETURN
,
,
PROCEDURE info
ALERT 3,"GFA BASIC|Mini-Banque_de_données",1," OK |egal ",back%
RETURN
,
PROCEDURE gerer
ALERT 1,"Gestion",1," OK ",back%
RETUR
,
PROCEDURE sauvegarder
ALERT 1,"Sauvegarde",1," OK ",back%
RETURN
,
PROCEDURE charger

```

```
    ALERT 1,"Chargement",1," OK ",back%  
RETURN  
,  
PROCEDURE afficher  
    ALERT 1,"Affichage/Impression",1," OK ",back%  
RETURN  
,  
PROCEDURE aider  
    ALERT 3,"Aide (s.o.s.)",1," OK ",back%  
RETURN  
,  
PROCEDURE traiter  
    ALERT 2,"Traitement",1," OK ",back%  
RETURN  
,  
PROCEDURE trier  
    ALERT 2,"Tri",1," OK ",back%  
RETURN  
,  
PROCEDURE rechercher  
    ALERT 2,"Recherche/Correction",1," OK ",back%  
RETURN
```

J'ai modifié un peu la formule des messages d'alarme, voyez vous-même quel effet cela peut avoir. J'espère que ceci vous incite à expérimenter par vous-même.

### Folding des procédures

Comme le texte de notre programme est devenu assez long, nous allons utiliser ici une possibilité offerte par la version 3.0 du GFA Basic : le folding. Placez votre curseur sur la ligne *PROCEDURE options* et appuyez sur la touche <Help>.

Toutes les lignes de la procédure ont brusquement disparu, et vous contemplez abasourdi ce qui reste, c'est à dire la ligne portant le nom de la procédure. Que s'est-il passé ? Que faire après le signe ">" ?

*Folding* signifie *replier* : le listing a été en quelque sorte replié de telle façon qu'on n'aperçoit plus que l'en-tête de la procédure. Amenez de nouveau votre curseur dessus et appuyez sur la touche <Help> : vous constatez que tout est redevenu comme avant.

Répétez ce processus de folding avec la procédure *choix* puis, pour vous entraîner, avec les autres procédures. Il ne vous reste plus qu'une liste des noms de procédures :

```

REM mini-banque de données en GEM
,
options
ON MENU GOSUB choix
OPENW 0
REPEAT
  ON MENU
  UNTIL MOUSEK=2 OR choix%=16
'touche droite de la souris ou point "fin" du menu
END
,
,
> PROCEDURE options
,
> PROCEDURE choix
,
,
> PROCEDURE info
,
> PROCEDURE gerer
,
> PROCEDURE sauvegarder
,
> PROCEDURE charger
,
> PROCEDURE afficher
,
> PROCEDURE aider
,
> PROCEDURE traiter
,
PROCEDURE trier
,
PROCEDURE rechercher

```

Voilà qui vous aide à y voir plus clair dans les programmes longs. Et lorsque vous avez besoin de savoir ce qui se cache derrière une procédure, il vous suffit d'ouvrir l'*enveloppe* en actionnant la touche <Help> . Lorsque votre curiosité est satisfaite, ou que vous avez procédé à des modifications, refermez tout simplement l'*enveloppe* en actionnant de nouveau la touche <Help> .

Si vous sauvegardez le texte de votre programme dans cet état, le programme prend bonne note des *pliures*, et, lors d'un chargement ultérieur, il vous reviendra aussi bien *plié* que lors de la sauvegarde.

## 11.4. Vous aimez les fenêtres ?

Avant d'en terminer avec ce chapitre, nous allons prendre connaissance de quelques instructions, vous permettant d'ouvrir et de refermer jusqu'à quatre fenêtres comme dans le GEM. Essayez tout de suite ce programme :

```
REM Windowtest GFA V 3.0
OPENW 0
FOR i& = 1 TO 4
  TITLEW #i&,"fenêtre no. " + STR$(i&)
  INFOW #i&,"ouvrir..."
  OPENW i&
  PAUSE 100
NEXT i&

FOR i& = 1 TO 4
  INFOW #i&,"...et fermer"
  TOPW #i&
  PAUSE 50
  CLOSEW i&
NEXT i&
CLOSEW 0
END
```

Vous connaissez déjà *OPENW 0*, qui permet d'exclure de la surface utilisable de l'écran la barre de menu GEM. *OPENW* permet aussi d'ouvrir des fenêtres semblables à celles que vous connaissez en tant qu'utilisateur du GEM lorsque vous voulez voir le contenu d'une disquette.

Si vous voulez entrer un texte sur la ligne de titre ou la ligne d'information d'une fenêtre, il faut que vous le fassiez avant d'ouvrir cette fenêtre, à l'aide des instructions *TITLEW* et *INFOW*. La pause qui suit ne sert qu'à vous laisser le temps de considérer avec satisfaction ces enfantillages.

L'instruction *TITLEW* permet de donner un nouveau titre à la fenêtre numéro *n* pour la version 2.0.

Dans la version 3.0, *TITLEW*[#]*no,n\$* a une autre définition. Cette fonction écrit le texte *A\$* dans la ligne la plus haute de la fenêtre.

L'instruction *INFOW* transmet à la fenêtre numéro *n* la nouvelle ligne d'information.

La deuxième partie du programme sert à réactiver cette fenêtre grâce à *TOPW*, qui n'existe pas dans le GFA Basic 2.0. Il serait possible d'ajouter encore un



OPENW avec un texte d'information différent. Après une pause pour le moins créative, on pourrait la fermer de nouveau par CLOSEW.

A la fin, l'écran reprend sa taille normale, et le spectacle GEM est terminé. Dans la version 2.0, n'écrivez pas le signe suivant TITLEW et INFOW. Votre programme aura l'allure suivante :

```
REM Windowtest GFA V 2.0
OPENW 0
FOR i% = 1 TO 4
  TITLEW i%,"fenêtre no. " + STR$(i%)
  INFOW i%,"ouvrir..."
  OPENW i%
  PAUSE 50
NEXT i%
,
FOR i% = 1 TO 4
  INFOW i%,"...et fermer"
  OPENW i%
  PAUSE 50
  CLOSEW i%
NEXT i%
CLOSEW 0
END
```

Je voudrais ici vous signaler que le nouveau GFA Basic contient une foule d'instructions et de fonctions vous permettant de réaliser des projets très élaborés et complexes avec le GEM. Toutefois, si vous voulez exploiter correctement cet ensemble de routines, il faut auparavant que vous connaissiez parfaitement les structures du GEM ainsi que le langage-système de votre Atari ST. Je vous renvoie donc à des ouvrages plus approfondis sur le GFA et le GEM. Vous pouvez aussi explorer la disquette GFA, qui contient des listings de démonstration concernant également la programmation GEM.

J'espère vous avoir incité à l'expérimentation concrète, mais vous ne devez pas attendre plus d'aide de ma part, car ceci dépasserait largement les limites de cet ouvrage destiné aux débutants. Je me borne à vous donner un dernier conseil : comme vous risquez fort de vous tromper lors de vos essais dans le GEM, vous devez absolument avant toute chose sauvegarder votre programme sur une disquette.

## 11.5. Résumé

Ce chapitre était un peu plus court que les précédents. Mais il vous a permis d'entrer plus avant dans les mystères du GEM.

Après avoir dimensionné un tableau suffisamment grand pour votre texte, vous le transmettez au gestionnaire GEM grâce à "MENU tableau\$()". Dans les menus déroulants, c'est le texte saisi en tout premier qui servira d'en-tête dans la barre de menu. La suite des chaînes doit se terminer par une chaîne vide ("").

Après l'affichage des options sur l'écran, *ON MENU GOSUB <procédure>* vous permet de créer un *interrupteur* qui vous amène dans une procédure de choix si vous cliquez sur l'un des points du menu.

ON MENU engendre une boucle interrogeant le statut de la souris : en appuyant la touche gauche, vous choisissez une option et en appuyant sur la touche droite (MOUSEK=2) vous sortez du programme.

Dès que vous avez sélectionné un point du menu grâce à la touche gauche, le texte apparaît en inversion-vidéo, et MENU OFF le ramène à son état normal.

MENU(0) contient l'indice du point du menu qui a été activé. Après avoir été attribué à une variable, *choix%* est mis en oeuvre et conduit à la procédure concernée.

Vous vous souvenez que *OPENW 0* permet de retrancher la barre de menu de la surface utilisable dans l'écran. Vous pouvez grâce à :

OPENW n		ouvrir et actualiser une fenêtre
CLOSEW n		fermer une fenêtre
TITLEW #n	(v 3.0)	donner un texte à la ligne de titre
INFOW #n	(v 3.0)	donner un texte à la ligne d'information
TOPW #n	(v 3.0)	actualiser une fenêtre
TITLEW n	(v 2.0)	donner un texte à la ligne de texte
INFOW n	(v 2.0)	donner un texte à la ligne d'information

Et n'oubliez pas que vous pouvez ouvrir au maximum quatre fenêtres.

## Chapitre 12

### Collecte des données

**V**ous voilà équipé de votre programme qui tourne sans problème, et ne comporte aucune faute de syntaxe. Cependant, il n'en sort rien. Cela ressemble à un restaurant qui n'est pas encore ouvert : sa carte des menus est bien imprimée et suspendue à l'entrée, mais ça ne vous donne pas encore à manger !

Il serait temps de nous tourner vers la confection des mets. Il faut d'abord penser aux ingrédients. Pour gérer et traiter des données, il nous faut quelques accessoires, dont nous allons parler dans ce chapitre.

#### 12.1. Une banque sans données

Lorsqu'on parle de banque, on pense à un endroit sûr pour conserver des valeurs. Une banque de données serait donc un endroit où l'on rassemblerait et conserverait des données. Selon moi, un programme chargé de gérer ces données devrait être conçu pour les protéger. Notre point de collecte est la disquette ; même si vous travaillez avec un disque dur, la disquette doit être votre outil de sauvegarde abritant vos données en toute sûreté.

Comme nous ne pouvons pas entrer nos données plus ou moins en vrac sur une disquette, nous devons tout d'abord nous fabriquer un classeur, un fichier ou quelque chose de semblable. Nous devons nous soucier d'avoir intégré dans notre programme un tableau d'espaces-mémoire assez grand pour y faire entrer nos données.

Mettons-nous d'accord sur une valeur maximale encore assez petite, car quelques données peu nombreuses suffiront pour tester un programme, et nous

épargnerons du temps en travaillant sur de petits ensembles de données. Plus tard, vous modifierez cette valeur en fonction de vos besoins.

DIM fiche\$(9)

Nous avons ainsi réservé un tableau pour 9 chaînes de caractères. Je pense qu'en fait vous aurez besoin de plus d'une ligne ou d'une chaîne par fichier ou par fiche, si bien que notre définition ne suffit pas, car elle ne réserve qu'une chaîne par unité de données. Le GFA Basic admet des chaînes de caractères si longues qu'elles peuvent occuper plusieurs écrans. De telles chaînes seraient bien difficiles à maîtriser, et nous disposons tout de même de possibilités plus confortables.

Les données peuvent être de nature très différentes : il serait donc souhaitable de différencier les tableaux prédéfinis. Prenons l'exemple d'un fichier de personnes.

Il nous faut au moins le nom, le prénom et l'adresse ; selon l'utilisation, on peut y ajouter les renseignements suivants :

téléphone, sexe, âge, profession, curriculum vitae, situation familiale, scolarité, situation financière, activités politiques ou associatives, casier judiciaire, état de santé etc.

Vous constatez qu'il n'y a aucune limite à la variété possible des types de données ! Limitons-nous à DIMensionner

Nom et Prénom  
Rue et numéro  
Code postal et bureau distributeur  
Téléphone avec son indicatif.

Grâce à :

DIM personne\$(9,4)

nous pourrions (en comptant à partir de 1) enregistrer 9 personnes, en réunissant sur une même ligne, le nom et le prénom, la rue et le numéro d'habitation, le code postal et le bureau distributeur, le téléphone et son indicatif. Vous préféreriez sans doute un autre découpage :

max% = 9  
DIM nom\$(max%),rue\$(max%)  
DIM adresse\$(max%),telephone\$(max%)

Dans ce cas, il est préférable d'attribuer une valeur maximale à une seule variable : si par la suite, vous voulez modifier la valeur de *max%* il vous suffira de modifier une seule valeur.

Il y a deux façons différentes de répartir la définition des données. Soit en réunissant toutes les données possibles dans un seul tableau même si elles sont de dimensions différentes. Soit en les répartissant de façon différenciée dans autant de tableaux qu'il y a de types de données différentes.

L'utilisateur d'un tel programme est en principe indifférent à la façon dont ses données sont organisées et gérées par le système interne. Il n'en va pas de même du programmeur, vous, qui voudrait fabriquer un programme clair et sans faute.

Admettons que nous voulons fabriquer un classeur ou un fichier contenant 9 fiches, ce qui nous suffira pour notre exemple : vous pourrez plus tard augmenter à volonté ces dimensions. Dans un premier cas, nous aurions pour ainsi dire un seul fichier avec des rayonnages qui contiendraient toutes les fiches :

```
DIM personne$(9,4)
```

Lors des tris ou des corrections, tout resterait ensemble, puisque toute procédure serait applicable à l'ensemble du fichier. L'écriture du programme d'entrée et de sortie de données nécessiterait certes une double boucle, mais la partie consacrée aux instructions serait courte :

```
REM tout en un
DIM personne$(9,4)
FOR i% = 1 TO 9
  FOR j% = 1 TO 4
    INPUT "notices individuelles";personne$(i%,j%)
    PRINT personne$(i%,j%)
  NEXT j%
NEXT i%
END
```

Dans le deuxième cas, nous aurons par contre plusieurs classeurs, plusieurs fichiers :

```
max% = 9
DIM nom$(max%)
DIM rue$(max%)
DIM adresse$(max%)
DIM telephone$(max%)
```

Ceux-ci peuvent se ranger dans une seule armoire ou être dispersés dans la pièce, leur ordre n'est pas encore indiqué. Lors des corrections et des tris, il faudra faire attention à ne pas tout mélanger, car les traitements devront intervenir dans tous les classeurs à la fois ! Pour l'entrée et la sortie des données, il faudra une longue suite d'instructions même si elle ne nécessite qu'une seule boucle :

```
REM chacun pour soi
max% = 9
'
DIM nom$(max%),rue$(max%)
DIM adresse$(max%),telephone$(max%)
'
FOR i% = 1 TO 9
  INPUT "Nom et Prénom : ",nom$(i%)
  PRINT nom$(i%)
  INPUT "Rue et numéro : ",rue$(i%)
  PRINT rue$(i%)
  INPUT "Code postal et bureau distributeur : ",adresse$(i%)
  PRINT adresse$(i%)
  INPUT "Téléphone avec indicatif : ",telephone$(i%)
  PRINT telephone$(i%)
NEXT i%
END
```

## 12.2. Abondance de données ?

La première méthode, consistant à tout réunir dans un seul fichier, paraît meilleure que la deuxième. Elle semble plus maniable, le programme est plus court. Mais avec l'augmentation et la différenciation croissantes des données, ce gros fichier va devenir impraticable.

Imaginez un peu tous les renseignements variés que vous pouvez avoir à réunir, et en plus au sujet de milliers d'individus, contenus dans un seul immense *classeur* !

Après l'instruction INPUT, le critère d'interrogation *notices individuelles* ne vous est pas d'un grand secours, et si vous voulez subdiviser ce critère, les deux variantes du programmes ne seront plus tellement différentes l'une de l'autre :

```
REM chacun le sien
DIM personne$(9,4)
'
FOR i% = 1 TO 9
  INPUT "Nom et Prénom : ",personne$(i%,1)
  PRINT personne$(i%,1)
```

```

INPUT "Rue et numéro : ",personne$(i%,2)
PRINT personne$(i%,2)
INPUT "Code postal et bureau distributeur : ",personne$(i%,3)
PRINT personne$(i%,3)
INPUT "Téléphone avec indicatif : ",personne$(i%,4)
PRINT personne$(i%,4)
NEXT i%
END

```

Tout est naturellement bien rangé maintenant, et chaque donnée est accessible séparément. Mais toutes ces données -pourtant si différentes l'une de l'autre- restent coiffées par un seul chapeau (personne\$). Si bien que pour distinguer entre les données, vous avez besoin de notes complémentaires, que vous pouvez consigner à la main sur une feuille de papier laissée à côté de votre ordinateur. Vous pouvez aussi intégrer dans votre programme quelques commentaires, ce qui accroît parfois considérablement la longueur du texte.

Ces commentaires sont superflus si, dans le cas idéal, toutes les constantes, variables et procédures portent des noms suffisamment significatifs, ne nécessitant que des commentaires très brefs. En tant que programmeur, vous pourrez lire et comprendre un tel programme même longtemps après son écriture, et il sera compréhensible facilement par d'autres programmeurs.

Il nous resterait donc la deuxième variante pour *emballer* nos données, c'est à dire fabriquer plusieurs petits classeurs, portant chacun un nom. Si on rapporte cela à notre exemple de renseignements sur des individus, il nous faudrait un classeur pour les noms et prénoms, un pour les adresses, bref il nous faudrait un classeur pour chaque type de données. Lorsque vous devrez effacer ou modifier des données sur une personne, il vous faudra faire attention à bien reporter toutes les modifications dans chacun des classeurs.

Visiblement, aucune des deux solutions n'est pleinement satisfaisante. Que pensez-vous du programme suivant :

```

RECORD personne$
  nom$(max%)
  rue$(max%)
  adresse$(max%)
  telephone$(max%)
ENDRECORD

```

Dans le cas de ce fichier défini par *RECORD* *personne\$* nous avons de nouveau affaire à un seul grand meuble, comme auparavant avec *DIM* *personne\$*. Mais ce meuble n'est plus équipé seulement de rayonnages pour abriter les données, il comporte des tiroirs étiquetés qui n'ont pas forcément les mêmes dimensions.

Vue de l'extérieur, la structure de RECORD est de type fermé : lors des tris ou des corrections, rien ne peut *s'échapper* ou se mélanger, avantage qu'avait déjà notre grand classeur unique ci-dessus, mais pas notre multitude de petits classeurs de la deuxième variante. Comme les classeurs, nos tiroirs sont étiquetés et offrent une vue d'ensemble claire pour le programmeur. On peut intervenir sur ses composants par :

```
personne$.nom$(i%)
personne$.rue$(i%)
personne$.adresse$(i%)
personne$.telephone$(i%)
```

Cela vous plaît ? C'est bien dommage car voilà une cruelle déception pour vous : le GFA Basic (même dans sa version 3.0) ne vous offre pas cette possibilité d'écriture. Je n'ai fait que l'emprunter au Pascal et l'adapter au Basic par de légères modifications. Pour moi, l'impossibilité de réunir en une seule structure des données de type différent reste une faiblesse du GFA Basic, pourtant si riche par ailleurs.

Il nous faut bien nous accommoder de ce petit défaut, et convenir d'une définition dès le début du programme principal, que vous pourrez bien sûr étendre ou raccourcir selon vos besoins :

```
REM mini banque de données
'
max%=9
'
DIM nom$(max%)
DIM rue$(max%)
DIM adresse$(max%)
DIM telephone$(max%)
'
no%=0
' autres instructions
'
END
'
' définitions des procédures
```

Nous utilisons, comme compteur des fiches de données, une variable globale "no%" (semblable à "choix\$"). Comme "max%", elle se compose d'un nombre entier, et vous permet de numérotter jusqu'à deux millions de fiches, ce qui devrait tout de même vous suffire. Au tout début, nous lui donnons la valeur "0", ce qui revient à dire *il n'y a pas de fichier ici* ou *toutes les fiches sont vides*.



## 12.3. Enregistrer !

Nous avons déjà vu et revu comment on entre ou sort des données simples, par exemple des chiffres, des mots, des phrases, bref du texte. Ici, il s'agit de grandes masses de données, qui peuvent avoir une structure plus ou moins complexe.

Naturellement, vous avez bien raison de penser que nous pourrions nous en tirer avec les instructions habituelles INPUT et PRINT. Je pense que vous serez d'accord avec moi pour dire que nous devrions mettre au point notre propre programme de régulation des données. Presqu'aucun programme - qu'il soit utilitaire ou de jeu - ne peut se passer d'une procédure pour l'entrée et la sortie des données, mais la façon de s'y prendre est très différente selon le but recherché :

Par exemple, dans un programme de traitement de texte, la saisie du texte et son affichage à l'écran sont étroitement liés, car on veut non seulement voir au fur et à mesure ce que l'on dactylographie, mais aussi rappeler les textes déjà saisis pour les modifier - rajouter un mot ici, en effacer un autre là.

Pour les jeux, l'entrée et la sortie forment une même unité indissociable : chaque mouvement de la manette de jeu (= entrée) est affiché immédiatement à l'écran (= sortie). Dans les jeux d'aventures, l'entrée et la sortie ressemblent plus à une sorte de jeu question/réponse ; une méthode utilisant différents programmes utilitaires. Vous constateriez qu'il est bien rare que la saisie soit intéressante si elle est dissociée de l'écran, surtout si votre saisie par les touches du clavier ne s'affichait brusquement plus au fur et à mesure sur votre écran.

Venons-en enfin à l'écriture d'un programme pour la saisie de nos données. Tout d'abord, remplaçons les différents points de la procédure de choix par l'appel de la procédure *enregistrer* :

Cette dernière va remplacer notre procédure *Traiter*.

```
PROCEDURE choix ! version-BOX
```

```
  :
```

```
  SELECT choix$
```

```
  :
```

```
  :
```

```
  CASE "6","T"
```

```
    enregistrer
```

```
  :
```

```
  :
```

```
RETURN
```

ou encore

```
PROCEDURE choix ! version-GEM
:
SELECT choix%
:
CASE 19
    enregistrer
:
RETURN
```

Nous reviendrons plus loin sur la procédure *traitement* ! Vous vous y risquez seul, ou vous préférez d'abord jeter un coup d'oeil sur la routine ?

```
PROCEDURE enregistrer
CLS
col&=4
LOCATE col&,5
INPUT "Nom      :",nom$(no%)
LOCATE col&,7
INPUT "Rue      :",rue$(no%)
LOCATE col&,9
INPUT "Adresse  :",adresse$(no%)
LOCATE col&,11
INPUT "Téléphone:",telephone$(no%)
RETURN
```

Si vous ne disposez que de la version 2.0 du GFA Basic, il faut remplacer toutes les instructions *LOCATE* <ligne>, <colonne> par *PRINT AT* (<colonne>, <ligne>);"";" et tous les & par %.

Nous nous permettons ici l'usage d'une nouvelle variable *col&* pour notre procédure *enregistre*, différente de *pos&* qui est réservée au menu. Pour la saisie, il nous manque encore une boucle, car il faut tenir compte des conditions suivantes lors de l'enregistrement des données et de leur répétition possible:

- a. on peut au plus entrer *max%* de données
- b. on doit pouvoir interrompre la saisie à volonté.

Essayez maintenant sans l'aide de ce manuel de formuler ces conditions en GFA Basic et de les insérer dans la boucle qui vous semble la plus appropriée. Vous ne lirez la solution qu'après avoir fini :

```
PROCEDURE enregistrer
LOCAL clavier$
:
```

```

col& = 4
CLS
WHILE clavier$ < > CHR$(27) AND no% < max%
  CLS
  INC no%
  LOCATE col&,5
  INPUT "Nom      : ",nom$(no%)
  LOCATE col&,7
  INPUT "Rue      : ",rue$(no%)
  LOCATE col&,9
  INPUT "Adresse   : ",adresse$(no%)
  LOCATE col&,11
  INPUT "Téléphone: ",telephone$(no%)
  LOCATE col&,13
  PRINT "< Esc > FIN < sinon > continuer"
  REPEAT
    clavier$ = INKEY$
  UNTIL clavier$ < > ""
WEND
RETURN

```

Toutes les fiches sur lesquelles on va écrire sont comptabilisées à l'intérieur de la boucle grâce à :

```
INC no%
```

Cette incrémentation ainsi que le bloc de saisie qui suit, sont répétés jusqu'à ce que la touche <ESC> soit actionnée :

```
clavier$ < > CHR$(27)
```

(il faut pouvoir interrompre la saisie à n'importe quel moment) et aussi longtemps qu'il reste des fiches disponibles :

```
no% < max%
```

(on peut entrer au maximum *max%* de données).

Je pense que le test à la fin de la boucle n'est pas très satisfaisant : si toutes les fiches de votre fichier sont remplies, et que vous lancez tout de même la procédure, *enregistrer* en tentant d'entrer des données, l'interpréteur GFA va vous répondre par le message *index de champ trop grand*.

D'ailleurs, si vous voulez demander le nombre d'éléments du tableau en cours d'écriture contenus dans votre programme, vous devez utiliser l'instruction *DIM ?* par exemple *DIM ?(telephone\$())*, ce qui vous permettra de savoir combien d'éléments le numéro de téléphone occupe .

## 12.4. Résumé

Vous avez dû avaler pas mal de théorie dans ce chapitre sans apprendre vraiment beaucoup de choses nouvelles. Vous en savez un peu plus sur les structures des données. Vous vous souvenez aussi qu'en GFA Basic vous pouvez déclarer des tableaux simples ou à plusieurs dimensions (arrays) grâce à

```
DIM <nom> (indice1,indice2...).
```

Vous obtenez le nombre total des éléments du tableau par

```
DIM ?(<nom> ).
```

Et vous devez vous accommoder du fait qu'en GFA Basic 3.0, il n'existe pas de variable RECORD contrairement au Pascal.

## Chapitre 13

### Manipulation des données

#### après formatage

Votre banque est ouverte, on peut y faire les premiers *versements*. Mais vous n'avez pas encore de formulaires pour cela. Il nous faut donc pour la représentation de nos données, élaborer un format ressemblant à une fiche, que nous pourrions aussi utiliser pour les *sorties de données* ; nous allons maintenant nous en occuper.

#### 13.1. Il nous faut un formulaire

Pour mettre nos données *en forme*, nous pouvons d'une part recourir aux instructions BOX ou RBOX. D'autre part, nous pourrions écrire une procédure reprenant tout le texte des noms des champs du fichier sous forme de *formulaire*, ce qui nous aiderait pour l'entrée des données :

```
PROCEDURE formulaire
  LOCAL form&
  ,
  form& = 4
  PRINT AT(form&,7);"Nom, Prénom....."
  PRINT AT(form&,11);"Rue....."
  PRINT AT(form&,15);"Adresse....."
```

```
PRINT AT(form&,19);"Téléphone....."
RETURN
```

Un peu pénible, non ? Nous n'aurions pas dû reprendre cela de la procédure d'enregistrement, car il va nous falloir ajouter une procédure pour l'affichage, pour lequel nous aurons de plus besoin d'un formulaire ! Vous avez bien raison, cette procédure est trop succincte, nous allons y ajouter des choses.

Fixons-nous d'emblée comme but d'écrire une *procédure formulaire* assez complète pour pouvoir servir dans d'autres programmes. Le texte dans sa forme concrète ne devrait donc pas figurer à l'intérieur de la procédure mais lui être transmis du dehors comme une variable. Comment faire cela ? Bricolons d'abord une routine de test, et nous verrons plus tard comment l'intégrer dans notre mini-banque de données :

```
REM testformulaire
DIM textform$(4)
'
FOR i& = 1 TO 4
  READ textform$(i&)
NEXT i&
'
DATA Nom et Prénom ....., Rue.....
DATA Adresse.....,Téléphone.....
```

Nous avons maintenant un tableau rempli avec le texte à afficher. Nous pouvons ensuite appeler la procédure *formulaire* après l'avoir adaptée en conséquence :

```
PROCEDURE formulaire
LOCAL form&,i&
LOCAL debut&
'
form& = 4
debut& = 7
FOR i& = 1 TO 4
  PRINT AT(form&,debut&);textform$(i&)
  ADD debut&,4
NEXT i&
RETURN
```

La procédure vous plaît maintenant ? Vous voyez que le curseur est replacé à chaque fois dans une boucle de comptage et qu'un nouvel élément du tableau *textform\$* est affiché à chaque fois. Ceci ne fonctionne dans notre exemple qu'à partir d'une position de début définie et pour quatre éléments. Nous pourrions rendre la procédure plus complète en incluant la position *debut&* et le nombre des éléments dans les paramètres à transmettre :

```

PROCEDURE formulaire(debut&,nombre&)
  LOCAL form&,i&
  LOCAL pas&
  ,
  form&=4
  pas&=(24-debut&) DIV (nombre&-1)
  FOR i&=1 TO nombre&
    PRINT AT(form&,debut&);textform$(i&)
    ADD debut&,pas&
  NEXT i&
  RETURN

```

## 13.2. La forme et le contenu

Nous touchons un peu aux mathématiques, comme le montre la ligne

```
pas&=(24-debut&) DIV (nombre&-1)
```

Pour déterminer l'écart entre les lignes de texte à afficher - le *pas* - il faut d'abord calculer combien il nous reste de lignes à l'écran pour l'affichage :

```
24-debut&
```

En admettant qu'on transmette pour *debut&* une valeur autorisée, il nous resterait au maximum 23 lignes après la ligne du début. Dans la première ligne utilisable viennent s'écrire les premières données saisies. Il nous reste alors *nombre&-1* lignes de texte, à répartir sur le reste de l'écran : *pas&* doit être un nombre entier arrondi, d'où l'utilité de l'opérateur DIV. Comme il est interdit de diviser par zéro, *nombre&* ne peut avoir la valeur 1, ce qui n'est pas grave pour l'utilisation de notre procédure *formulaire*. Mais il reste à savoir s'il nous faudra une procédure spéciale pour afficher uniquement une ligne de texte !

Vous pouvez rendre votre procédure-formulaire absolument sûre en faisant tester au début de la procédure tous les paramètres interdits et en les transformant en la valeur autorisée la plus approchante. Je n'insisterai pas plus là-dessus, et je vous laisse le soin de terminer ce travail.

L'autre possibilité consisterait à intégrer directement dans la procédure l'instruction READ, ce qui rendrait superflue la déclaration du tableau *textform\$* dans le programme principal :

```

PROCEDURE formulaire(debut&,nombre&) ! READ
LOCAL form&,i&
LOCAL pas&
'
form&= 4
pas&= (24-debut&) DIV (nombre&-1)
FOR i&= 1 TO nombre&
  READ textform$
  PRINT AT(form&,debut&);textform$(i&)
  ADD debut&,pas&
NEXT i&
RESTORE
RETURN

```

Cela vous plaît peut-être plus ? Naturellement, il faut que les lignes DATA correspondantes existent bien. Pour le cas où vous voudriez utiliser plusieurs fois la procédure-formulaire dans votre programme, il faut que le pointeur DATA qui se trouve à la fin de la dernière ligne DATA après la lecture provoquée par READ, soit remplacé au début de la première ligne grâce à l'instruction RESTORE.

Si vous voulez intégrer sous cette forme la procédure dans un programme contenant plusieurs instructions DATA, il vous faut de plus, caractériser chaque bloc DATA en lui attribuant une marque (label). Le pointeur utilisé pour la procédure *formulaire* peut alors retrouver le début du premier élément par *RESTORE <marque>*. Sinon il se retrouvera toujours devant la première ligne DATA qu'il rencontrera à partir du début du programme.

La commande RESTORE devrait de plus figurer tout au début, avant l'instruction READ, afin de pouvoir lire immédiatement à partir du bon endroit. En résumé, vous obtiendrez le programme suivant, dans lequel naturellement les procédures et les lignes DATA peuvent se trouver à différents endroits :

```

PROCEDURE formulaire(debut&,nombre&) ! READ marque
LOCAL form&,i&
LOCAL pas&
'
RESTORE textform
form&= 4
pas&= (24-debut&) DIV (nombre&-1)
FOR i&= 1 TO nombre&
  READ textform$(i&)
  PRINT AT(form&,debut&);textform$(i&)
  ADD debut&,pas&
NEXT i&
'
TEXTFORM :
DATA Nom et Prénom ...., Rue.....

```



```
DATA Adresse.....,Téléphone.....,
RETURN
```

Naturellement, votre oeil perçant a tout de suite remarqué que j'ai glissé là-dedans une virgule de plus, ce qui me donne 5 éléments DATA, le dernier étant un *dummy* (un élément *pour rien*), car sinon la présentation avec les cadres suivants ne serait plus si jolie, ce qui serait bien dommage, non ?

Attention, modifiez la déclaration du tableau textform\$ par DIM textform\$(5)

### 13.3. L'entrée des données...

Décidez-vous pour l'une ou l'autre version, car nous allons continuer avec l'insertion de cette procédure formulaire dans la routine de saisie des données qui doit elle aussi recevoir un petit cadre :

```
PROCEDURE enregistrer
LOCAL i&
LOCAL clavier$
'
' masque de saisie
CLS
RBOX 8,60,631,340          lbord de la fiche
RBOX 8,8,631,40            len-tête
RBOX 8,344,631,375         lbas de la fiche
FOR i& = 5 TO 17 STEP 4
  RBOX 16,16*i&,623,16*i& + 48  ltextform$
NEXT i&
'
' saisie des données
col& = 25
WHILE clavier$ < > CHR$(27) AND no% < max%
  INC no%
  PRINT AT(4,2);"fiche no ";no%
  formulaire(7,5)
  LOCATE col&,7
  INPUT "",nom$(no%)
  LOCATE col&,11
  INPUT "",rue$(no%)
  LOCATE col&,15
  INPUT "",adresse$(no%)
  LOCATE col&,19
  INPUT "",telephone$(no%)
  PRINT AT(4,23);" < Esc > FIN < sinon > continuer";
  REPEAT
    clavier$ = INKEY$
  UNTIL clavier$ < > ""
```

```
WEND
RETURN
```

Vous pouvez maintenant enregistrer vos premières données, pour que votre banque ne soit plus aussi vide. Vous ne semblez toujours pas très heureux : à chaque fois que vous changez de fiche, vous conservez à l'écran les jolis petits cadres mais hélas aussi toutes les données que vous venez de saisir sur la fiche précédente. Nous nous attendions à recevoir de nouvelles fiches toute *neuves*, et nous nous retrouvons avec des fiches déjà écrites qu'il faut raturer, ce qui est pire que tout !

La plus radicale des solutions consisterait à provoquer, avant la reprise du masque de saisie, l'effacement complet de l'écran et l'appel d'un nouveau masque. Mais ceci vous ferait perdre du temps. Il est beaucoup plus élégant de ne vider que les zones à compléter en gardant le masque, ce que fera votre procédure formulaire si nous la modifions un peu :

```
PROCEDURE formulaire(debut&,nombre&)
LOCAL form&,i&
LOCAL pas&,rest&
,
RESTORE textform
form& = 4
pas& = (24-debut&) DIV (nombre&-1)
FOR i& = 1 TO nombre&
  READ textform$(i&)
  rest& = 50-LEN(textform$)
  PRINT AT (form&,debut&);textform$(i&);SPACE$(rest&)
  ADD debut&,pas&
NEXT i&
,
textform :
DATA Nom et Prénom ....., Rue.....
DATA Adresse.....,Téléphone.....,
RETURN
```

Nous avons ajouté la variable *rest&* : si on prend 50 caractères comme longueur de ligne, il nous reste après l'affichage de *textform\$* :

```
rest& = 50-LEN(textform$)
```

La fonction *SPACE\$* va remplir ce *rest&* d'espaces vides, ce qui *gommera* tout ce qui était inscrit à cet endroit auparavant. Naturellement la gomme ne fonctionne que pour l'écran, votre ordinateur lui n'oubliera pas les données entrées.

Vous avez peut-être remarqué la gestion *géniale* de la ligne de bas de fiche, qui disparaît durant la saisie des données, et réapparaît pour vous signaler qu'une fiche est remplie. Vous pouvez enfin *jouer avec des fiches propres* et entrer vos données sans problème.

Nous allons profiter de cette fin de section pour vous donner le listage complet de notre programme réalisé jusqu'ici.

```

REM NIMI-BANQUE DE DONNÉES
DIM textform$(5)
'
' FOR i& = 1 TO 4
' READ textform$(i&)
' NEXT i&
'
'
max% = 9
'
DIM nom$(max%)
DIM rue$(max%)
DIM adresse$(max%)
DIM telephone$(max%)
'
no% = 0
options
BOX 88,42,528,328
BOX 96,50,320,92
BOX 320,50,520,92
BOX 96,100,320,220
BOX 320,100,520,220
BOX 96,228,320,268
BOX 320,228,520,268
BOX 88,292,528,328
REPEAT
  choix
UNTIL choix$ = "0" OR choix$ = CHR$(27)
'
PROCEDURE options
  pos& = 15
  PRINT AT(23,5);"FICHIER"
  PRINT AT(50,5);"DONNEES"
  PRINT AT(pos&,8);"1. Gestion"
  PRINT AT(43,8);"6. Enregistrement"
  PRINT AT(pos&,10);"2. Sauvegarde"
  PRINT AT(43,10);"7. trl"
  PRINT AT(pos&,11);"3. Chargement"
  PRINT AT(43,11);"8. Recherche/Remplace"
  PRINT AT(pos&,13);"4. Affichage/Impression"
  PRINT AT(pos&,16);"5. aide(s.O.s)"
  PRINT AT(43,16);"0. Fin"
  PRINT AT(pos& + 8,20);" -appuyez sur une touche S.V.P.-"
RETURN
'

```

```

PROCEDURE choix
REPEAT
  choix$ = UPPER$(INKEY$)
UNTIL choix$ < > ""
SELECT choix$
CASE "0",27
  REM VIDE
CASE "1","G"
  gerer
CASE "2","S"
  sauvegarder
CASE "3","C"
  charger
CASE "4","A"
  afficher
CASE "5","O"
  aider
CASE "6","T"
  enregistrer
CASE "7","I"
  trier
CASE "8","R"
  rechercher
DEFAULT
  choix
ENDSELECT
RETURN
,

PROCEDURE gerer
  ALERT 1,"gestion",1," OK ",back%
RETURN
,

PROCEDURE sauvegarder
  ALERT 1,"sauvegarde",1," OK ",back%
RETURN
,

PROCEDURE charger
  ALERT 1,"chargement",1," OK ",back%
RETURN
,

PROCEDURE afficher
  ALERT 1,"affichage/impression",1," OK ",back%
RETURN
,

PROCEDURE aider
  ALERT 3,"aide (S.O.S)",1," OK ",back%
RETURN
,

PROCEDURE enregistrer
  LOCAL i&
  LOCAL clavier$
  ,
  ' masque de saisie
  CLS
  RBOX 8,60,631,340

```

```

RBOX 8,8,631,40
RBOX 8,344,631,375
FOR i&=5 TO 17 STEP 4
  RBOX 16,16*i&,623,16*i&+48
NEXT i&
'saisie des données
col&=25
WHILE clavier$ < > CHR$(27) AND no% < max%
  INC no%
  PRINT AT(4,2);"fiche no ";no%
  formulaire(7,5)
  LOCATE col&,7
  INPUT "",nom$(no%)
  LOCATE col&,11
  INPUT "",rue$(no%)
  LOCATE col&,15
  INPUT "",adresse$(no%)
  LOCATE col&,19
  INPUT "",telephone$(no%)
  PRINT AT(4,23);" < ESC > FIN < sinon > continuer"
  REPEAT
    clavier$ = INKEY$
  UNTIL clavier$ < > ""
WEND
RETURN
'
PROCEDURE trier
  ALERT 2,"Trier",1," OK ",back%
RETURN
'
PROCEDURE rechercher
  ALERT 2,"recherche/correction",1," OK ",back%
RETURN
'
PROCEDURE formulaire(debut&,nombre&)
  LOCAL form&,i&
  LOCAL pas&,rest&
  '
  RESTORE textform
  form&=4
  pas&=(24-debut&)/DIV (nombre&-1)
  FOR i&=1 TO nombre&
    READ textform$(i&)
    rest&=50-LEN(textform$)
    PRINT AT(form&,debut&);textform$(i&);SPACE$(rest&)
    ADD debut&,pas&
  NEXT i&
  textform:
  DATA Nom et Prénom.....,Rue.....
  DATA Adresse.....,Téléphone.....
RETURN

```

## 13.4. ... et la sortie des données

Vous n'êtes toujours pas content ? Vous avez certainement commis des fautes en saisissant les données, et vous voilà fâché de ne pas pouvoir les corriger. Une procédure de correction serait bien utile, non ? Prenez un peu patience, vous n'en êtes qu'à la phase de test : il est à ce stade peu important pour l'ordinateur comme pour votre programme que vous ayez commis quelques erreurs de saisie !

Je pense qu'il est beaucoup plus important de chercher maintenant à visualiser les données déjà entrées. Contrôlez avant tout si votre procédure de choix vous affiche bien la routine de sortie suivante :

```
PROCEDURE choix ! version-BOX
```

```
  :  
  SELECT choix$  
  :  
  :  
  CASE "4","A"  
    afficher  
  :  
  :  
  RETURN
```

ou encore:

```
PROCEDURE choix ! version-GEM
```

```
  :  
  SELECT choix%  
  :  
  :  
  CASE 14  
    afficher  
  :  
  :  
  RETURN
```

N'oubliez pas d'effacer d'abord la procédure correspondante existant sous ce nom ! Vous devriez d'ailleurs être en mesure de fabriquer vous-même la procédure d'affichage, en examinant soigneusement la procédure *enregistrer* et en reprenant les instructions pour le formulaire et les cadres :

```
PROCEDURE afficher  
  LOCAL i&  
  LOCAL clavier$,aff%  
  ,  
  'masque d'affichage  
  CLS
```

```

RBOX 8,60,631,340 ! bord de la fiche
RBOX 8,8,631,40 ! en-tête
RBOX 8,344,631,375 ! bas de la fiche
FOR i& = 5 TO 17 STEP 4
  RBOX 16,16*i&,623,16*i& + 48 ! textform$
NEXT i&
'
' affichage des données
col& = 25
WHILE clavier$ < > CHR$(27) AND aff% < no%
  INC aff%
  PRINT AT(4,2);"fiche no ";aff%
  formulaire(7,5)
  LOCATE col&,7
  PRINT """,nom$(aff%)
  LOCATE col&,11
  PRINT """,rue$(aff%)
  LOCATE col&,15
  PRINT """,adresse$(aff%)
  LOCATE col&,19
  PRINT """,telephone$(aff%)
  PRINT AT(4,23);" < Esc > FIN < sinon > continuer";
  REPEAT
    clavier$ = INKEY$
  UNTIL clavier$ < > ""
WEND
RETURN

```

Vous comprenez bien que la procédure d'affichage doit posséder sa propre variable locale *aff%*, puisque *no%* renvoie à la dernière fiche écrite, ce qui fait que *no%* représente la valeur maximum pour l'affichage car toutes les autres fiches sont encore vierges. La procédure n'est pas très différente de la procédure *enregistrer*.

Vous le saviez ? et vous vous souvenez encore des possibilités offertes par l'éditeur GFA pour le traitement des blocs ? Alors vous savez comment vous épargner ici du travail d'écriture.

Vous marquez le début puis la fin de la procédure *enregistrer* en cliquant sur *BlkSta* ou en appuyant < shift > < F5 > puis *BlkEnd* ou < F5 >. Vous amenez ensuite votre curseur là où vous voulez insérer ce bloc. Vous repassez dans le menu des blocs par *Block* ou < F4 >. Si vous n'avez pas marqué correctement votre bloc, vous voyez s'afficher : *Block ???*, il vous faut alors recommencer le marquage. A partir du menu des blocs, vous pouvez copier votre bloc marqué à l'aide de *Copy*, et supprimer ensuite le marquage par *Hide*.

Il suffit alors de remplacer *no* par *aff%* et *INPUT* par *PRINT*. Là-aussi vous pouvez vous épargner du travail : amenez d'abord le curseur au début du nouveau bloc à traiter, puis actionnez *Replace* ou < Shift > < F6 >.

Dans le GFA Basic 3.0, cela s'appelle *Replac* car le texte du menu y est davantage resserré. La fenêtre de l'éditeur vous invite tout en haut par *Find* : à entrer le texte que vous voulez modifier. Tapez ici INPUT puis <Return>.

Ensuite, la mention *replace* : vous demande par quoi vous voulez remplacer l'ancien texte. Entrez ici PRINT et terminez par <Return>. Il ne se passe tout d'abord rien. Le curseur se place sur le premier INPUT rencontré mais il faut actionner

<Control>-E (=remplacer en GFA Basic 3.0)

ou

<Control>-R (=replace en GFA Basic 2.0)

pour procéder au remplacement par PRINT. Vous allez ensuite au INPUT suivant par :

<Control>-E (GFA Basic 3.0)  
<Control>-F (GFA Basic 2.0)

Vous continuez alors ce petit jeu en répétant constamment <Control>-E ou <Control>-F et -R jusqu'à ce qu'il n'y ait plus rien à remplacer. Vous procédez de même pour remplacer *no%* par *aff%*. Les autres modifications, vous les faites *à la main*. Pour rechercher quelque chose dans le texte du programme, utilisez Find ou <F6>. Reportez-vous à l'annexe A qui récapitule toutes les options de l'éditeur.

## 13.5. Chirurgie plastique

Puisque vous en êtes aux modifications : ne vous êtes-vous pas fâché de voir disparaître le prénom de vos amis ? Ou avez-vous tout simplement renoncé à mettre une virgule dans *Dupont, Caroline* ? Vous n'y êtes pas obligé si vous utilisez LINE INPUT :

```
PROCEDURE enregistrer
:
:
: saisie des données
:
: formulaire(7,5)
LOCATE col&,7
LINE INPUT "",nom$(no%)
```



```

LOCATE col&,11
LINE INPUT "",rue$(no%)
LOCATE col&,15
LINE INPUT "",adresse$(no%)
LOCATE col&,19
LINE INPUT "",telephone$(no%)
:
RETURN

```

Vous n'avez plus à économiser les virgules, mais attention, **LINE INPUT** ne s'utilise qu'avec des chaînes de caractères. Cette limitation ne nous pose pas de problème dans notre mini-banque de données, car nous ne travaillons qu'avec des chaînes de caractères. Vous pouvez donc par la suite utiliser **LINE INPUT** pour *nom virgule prénom*...

J'aimerais aussi améliorer quelque chose sur les bords de la fiche : la ligne du bas me plairait plus dans les routines d'entrée et de sortie, si la mention *<sinon> continuer* se trouvait tout à fait à droite. Comme cette ligne s'écrit les deux fois, exactement de la même façon, on pourrait l'inclure dans la procédure suivante :

```

PROCEDURE bas (VAR clavier$)
PRINT AT(4,23);" < Esc >  FIN";SPACE$(48);" < sinon > continuer";
REPEAT
  clavier$ = INKEY$
UNTIL clavier$ < > ""
RETURN

```

Le paramètre *clavier\$* doit être défini comme variable globale afin que la procédure renvoie la valeur *clavier\$*. Dans la version 2.0, on n'utilise aucun paramètre, *clavier\$* a alors une portée globale.

La ligne d'en-tête me paraît surchargée sur sa gauche car elle est vide à droite. On pourrait y placer le mode de traitement, ce qui nous amène à cette routine :

```

PROCEDURE tete(nombre%,mode$)
PRINT AT(4,2);"fiche no ";nombre%
PRINT AT(60,2);mode$
RETURN

```

Si vous voulez, vous pouvez plutôt faire figurer sur cette ligne l'heure et la date par les fonctions **DATE\$** et **TIME\$** :

```
PROCEDURE tete(nombre%,mode$)
PRINT AT (4,2);"fiche no ";nombre%
PRINT AT (30,2);DATE$;" ";TIME$;" heure"
PRINT AT (60,2);mode$
RETURN
```

Enfin, je ne vois pas pourquoi on ne reprendrait pas dans une seule procédure, le même masque pour l'affichage et la saisie avec les instructions RBOX :

```
PROCEDURE masque
LOCAL i&
CLS
RBOX 8,60,631,340 !bord de la fiche
RBOX 8,8,631,40 !en-tête
RBOX 8,344,631,375 !bas de la fiche
FOR i&=5 TO 17 STEP 4
RBOX 16,16*i&,623,16*i&+48 ltextform$
NEXT i&
RETURN
```

Et voyez maintenant toutes les modifications :

```
PROCEDURE enregistre
LOCAL clavier$
,
masque
col&=25
WHILE clavier$ <> CHR$(27) AND no% < max%
INC no%
tete(no%,"saisie")
formulaire(7,5)
LOCATE col&,7
LINE INPUT "",nom$(no%)
LOCATE col&,11
LINE INPUT "",rue$(no%)
LOCATE col&,15
LINE INPUT "",adresse$(no%)
LOCATE col&,19
LINE INPUT "",telephone$(no%)
bas(clavier$)
WEND
RETURN
,
PROCEDURE afficher
LOCAL clavier$,aff%
,
masque
col&=25
WHILE clavier$ <> CHR$(27) AND aff% < no%
INC aff%
tete(aff%,"affichage")
```

```
formulaire(7,5)
PRINT AT(col&,7);nom$(aff%)
PRINT AT(col&,11);rue$(aff%)
PRINT AT(col&,15);adresse$(aff%)
PRINT AT(col&,19);telephone$(aff%)
bas(clavier$)
WEND
RETURN
```

J'ai réduit encore la procédure d'affichage en remplaçant LOCATE et PRINT par une instruction PRINT AT. Vous voilà maintenant saisi par l'envie de voir s'il n'y a pas d'autres points communs qu'on pourrait rassembler dans une procédure ? Je vais vous présenter une commande pour terminer : LPRINT.

Le "L" devant PRINT commande à l'ordinateur de sortir les données non-pas vers l'écran mais vers l'imprimante (lineprinter). Cela n'est pas très nouveau pour vous si vous vous souvenez de la signification de Llist.

Si vous possédez une imprimante, vous pouvez vous faire le plaisir -ou vous donner la peine- de convertir la procédure d'affichage pour en faire une procédure d'impression. Limitez-vous d'abord à l'impression brute des données. Réfléchissez à la façon d'insérer ici les procédures d'en\_tête et de bas-de-fiche. Vous pourriez élaborer le menu suivant :

Affichage
1. sur l'écran
2. sur l'imprimante

Vérifiez que votre imprimante est bien allumée avant d'appeler la routine d'impression.

## 13.6. Résumé

Fin de ce chapitre : je vous accorde une pause, après laquelle vous devrez vous souvenir de tel ou tel point traité dans les chapitres précédents.

N'oubliez pas que les procédures définies par PROCEDURE ne fonctionnent que si vous les appelez quelque part par leur nom, par GOSUB ou par un "@" placé devant !

Vous vous rappelez que "SPACE\$( < nombre > )" sert à produire un certain nombre d'espaces vides et DATE\$, TIME\$ à indiquer ou régler la date et l'heure.

Je vous ai rappelé READ et DATA et fait connaître la nouvelle commande RESTORE, qui ramène le pointeur DATA au début des lignes DATA. Si vous voulez le ramener à une place déterminée, il faut la marquer et compléter la commande en écrivant "RESTORE < marque >".

Vous savez que vous pouvez entrer des virgules si vous utilisez "LINE INPUT < variable-chaîne > ", et que vous pouvez lancer une impression sur imprimante par LPRINT, si cette dernière est allumée !

Je vous rappelle enfin ces commandes-éditeur :

BlkSta	< Shift > < F5 >	marquer le début d'un bloc
BlkEnd	< F5 >	marquer la fin d'un bloc
Block	< F4 >	accès au menu des blocs :
- Copy	< C >	-copier le bloc
- Move	< M >	-déplacer le bloc
- Write	< W >	-sauvegarder le bloc sur une disquette
- List	< L >	-imprimer le bloc sur imprimante
- Delete	< Control > < D >	-effacer le bloc
- Hide	< H >	-ôter les marques du bloc
Find	< F6 >	Retrouver le texte d'un bloc
Replac(e)	< Shift > < F6 >	Remplacer le texte

Les deux dernières options ne fonctionnent que si vous utilisez en plus < Control > -F et < Control > -R ou < Control > -E. Pour le reste de ce qui concerne l'éditeur, reportez-vous à l'annexe A.

Vous ne m'en voudrez pas de vous rappeler que vous pouvez *plier* le listing de votre programme en plaçant votre curseur sur PROCEDURE... et en appuyant sur la touche < Help > . On peut procéder à des opérations de bloc même sur des procédures *repliées* mais pas pour Find et Replac(e).

## Chapitre 14

### Utilisation des disquettes

**V**ous êtes bien avancé maintenant dans la réalisation de votre banque de données. Vous pouvez y faire des *versements* et des *retraits* pendant les heures d'ouverture. Mais vous aimeriez bien que votre compte soit bien à l'abri, même après les heures d'ouverture !

Il faut donc tout placer dans un *coffre-fort* avant de fermer les bureaux : je l'ai déjà dit, notre coffre-fort, c'est la disquette. Même si vous avez un disque dur, il est bon d'avoir une sauvegarde sur disquette.

#### 14.1. Ouvrir et fermer

Nous allons maintenant voir comment vous allez transporter sur la disquette vos classeurs et fichiers. Jusqu'ici, vous n'avez que sauvegardé vos programmes vers une disquette ou vers un fichier sur votre disque dur, à l'aide de Save ou Save,A, et vous les avez rechargés par Load ou Merge dans l'éditeur.

Il est clair que sauvegarder revient, pour l'ordinateur, à écrire et charger revient à lire, ce qui nous amène à penser à PRINT et INPUT. Nous ne les avons encore utilisés que pour les choses suivantes :

PRINT pour l'affichage sur l'écran

et

INPUT pour l'entrée de données par le clavier.

L'ordinateur doit maintenant recevoir un signe lui indiquant que cette fois ces instructions ne concernent pas l'écran ou le clavier :

```

REM écrire 5 phrases sur disquette
FOR i% = 1 TO 5
  INPUT phrase$      ! saisie par le clavier
  PRINT #1,phrase$   ! sortie sur une disquette
NEXT i%
END

REM lire 5 phrases sur disquette
FOR i% = 1 TO 5
  INPUT #1,phrase$    ! chargement depuis la disquette
  PRINT phrase$       ! affichage sur l'écran
NEXT i%
END
    
```

Le "#" figurant derrière PRINT ou INPUT est le signal destiné à faire comprendre à l'interpréteur GFA qu'il ne s'agit plus des entrées/sorties ordinaires. Vous supposez à juste titre qu'on peut alors solliciter d'autres appareils, comme par exemple l'imprimante. Le chiffre qui suit le "#" indique le canal de sortie, et correspond au numéro de l'appareil périphérique connecté sur ce port, ici l'unité de disquette. Ce nombre n'est pas forcément 1.

Les données sont alors tout simplement *déviées*. Si vous avez essayé de lancer un des deux exemples, vous avez dû récolter un message d'erreur, car tout est en fait plus compliqué qu'il n'y paraît. Il faut tout d'abord ouvrir le canal en question puis le fermer.

La définition-standard de tout ordinateur, de tout système, de tout langage de programmation prévoit toujours l'ouverture des canaux correspondant au clavier et à l'écran. Les autres canaux doivent par contre être expressément ouverts puis refermés. Les commandes du GFA Basic pour cela sont toutes simples :

OPEN.,,

et

CLOSE

L'instruction OPEN ouvre un canal de données ou un canal vers un fichier sur disquette.

"O"	ouvre un fichier en écriture
"I"	ouvre un fichier en lecture
"A"	permet d'ajouter des données à un fichier
"U"	ouvre un fichier existant en écriture et en lecture
"R"	ouvre un fichier à accès direct, fichier décrit sous l'instruction FIELD

L'instruction CLOSE ferme un canal de données ou un canal lié à un fichier sur disquette. Les 2 instructions vont de pair.

En ouvrant un canal, l'ordinateur veut savoir s'il va écrire (= "OUTPUT" abréviation "O") ou lire (= "INPUT" abréviation "I"). Comme il s'agit ici d'un disque dur ou d'une disquette, ces indications ne suffisent pas encore. Pour afficher ou entrer des données par le clavier, l'écran ou l'imprimante, il suffit d'un code désignant l'appareil visé. Sur une disquette ou un disque dur, on peut par contre accumuler une foule de fichiers différents qu'il faut appeler par leurs propres noms !

C'est pourquoi il faut préciser non-seulement le nom de l'unité de disque ou de disquette mais aussi le nom des fichiers, ce qui nous donnerait un programme-test comme celui-ci :

```
REM écrire sur disquette
DIM phrase$(5)
INPUT "nom du fichier : ",nomfichier$
OPEN "O",#1,nomfichier$
FOR i%= 1 TO 5
  INPUT phrase$(i%)      ! saisie par le clavier
  PRINT #1,phrase$(i%)   ! sortie sur une disquette
NEXT i%
CLOSE #1                 ! fermeture du fichier
'
'
REM lire la disquette
INPUT "nom du fichier : ",nomfichier$
OPEN "I",#1,nomfichier$
FOR i%= 1 TO 5
  INPUT #1,phrase$(i%)   ! chargement depuis la disquette
  PRINT phrase$(i%)      ! affichage sur l'écran
NEXT i%
CLOSE #1                 ! fermeture du fichier
END
```

*nomfichier\$* est le nom du fichier sous lequel le fichier sera sauvegardé et rechargé sur la disquette.

```
OPEN "O",#1,nomfichier$  (GFA 3.0 abr. o o 1 nomfichier$)
                        (GFA 3.0 abr. O "O" 1 nomfichier$)
```

permet d'ouvrir le fichier pour écrire dedans. S'il existait auparavant un fichier de même nom, il est écrasé.

```
OPEN "I",#1,nomfichier$  (GFA 3.0 abr. o i 1 nomfichier$)
                        (GFA 3.0 abr. O "I" 1 nomfichier$)
```

permet d'ouvrir un fichier pour le lire. Si le fichier demandé n'existe pas, vous recevez un message d'erreur, et les données entrées sont perdues !

J'espère que vous avez testé tout cela avec des données que vous pouviez perdre sans regret ! Vous en êtes sans doute arrivé à la conclusion qu'il faut ici se montrer particulièrement prudent !

Il vaut mieux s'assurer d'abord de la présence du fichier demandé sur la disquette placée dans l'unité de disquette.

Le GFA Basic va vous y aider : la fonction `EXIST` (nom de fichier\$) permet de savoir si tel ou tel fichier existe (valeur `TRUE` = -1) ou non (valeur `FALSE` = 0). Nous pouvons compléter notre programme :

```

REM écrire sur disquette
DIM phrase$(5)
INPUT "nom du fichier à écrire :", nomfichier$
IF NOT EXIST(nomfichier$)
  OPEN "O", #1, nomfichier$ ! fichier n'existe pas encore
  FOR i% = 1 TO 5
    INPUT phrase$(i%) ! saisie par le clavier
    PRINT #1, phrase$(i%) ! sortie sur une disquette
  NEXT i%
  CLOSE #1 ! fermeture du fichier
ELSE
  PRINT "fichier existe déjà"
ENDIF
REM lire la disquette
INPUT "nom du fichier à lire : ", nomfichier$
IF EXIST(nomfichier$)
  OPEN "I", #1, nomfichier$ ! fichier existe déjà
  FOR i% = 1 TO 5
    INPUT #1, phrase$ ! chargement depuis la disquette
    PRINT phrase$(i%) ! affichage sur l'écran
  NEXT i%
  CLOSE #1 ! fermeture du fichier
ELSE
  PRINT "fichier non trouvé"
ENDIF
END
    
```

## 14.2. Un coffre-fort pour la banque

Vous voulez maintenant enfin posséder un coffre-fort pour votre banque. Cela ne devrait plus être bien difficile, puisque vous savez que les deux manipulations nécessaires s'apparentent au fond aux procédures d'entrée et



de sortie de données, qui se font cette fois non-plus avec le clavier et l'écran mais avec l'unité de disquette.

Dépouillons ces deux procédures de leurs masques de formulaire et de leurs cadres ainsi que des instructions LOCATE. On peut aussi retailler un peu dans les PRINT et INPUT.

Les lignes d'en-tête et de bas de fiche sont inutiles, et on peut aussi supprimer toutes les définitions et instructions concernant *clavier\$*. Nous obtenons donc deux *squelettes* sur lesquels nous pouvons greffer nos nouvelles procédures :

```
PROCEDURE afficher
LOCAL aff%
,
WHILE aff% < no%
  INC aff%
  PRINT nom$(aff%)
  PRINT rue$(aff%)
  PRINT adresse$(aff%)
  PRINT telephone$(aff%)
WEND
RETURN
,
,
PROCEDURE enregistrer
,
WHILE no% < max%
  INC no%
  LINE INPUT nom$(no%)
  INPUT rue$(no%)
  INPUT adresse$(no%)
  INPUT telephone$(no%)
WEND
RETURN
```

Il ne nous reste plus qu'à les adapter pour le travail avec l'unité de disquette :

```
PROCEDURE sauvegarder
LOCAL aff%
,
LOCATE 23,15
INPUT "nom du fichier :", nomfichier$
IF nomfichier$ < > ""
  OPEN "O", #1, nomfichier$
  WHILE aff% < no%
    INC aff%
    PRINT #1, nom$(aff%)
    PRINT #1, rue$(aff%)
    PRINT #1, adresse$(aff%)
    PRINT #1, telephone$(aff%)
  WEND
```

```

CLOSE #1
ENDIF
RETURN
'
'

PROCEDURE charger
'
LOCATE 23,15
INPUT "nom du fichier :",nomfichier$
IF EXIST(nomfichier$)
OPEN "I",#1,nomfichier$
WHILE no% < max%
INC no%
LINE INPUT #1, nom$(no%)
INPUT #1, rue$(no%)
INPUT #1, adresse$(no%)
INPUT #1, telephone$(no%)
WEND
CLOSE #1
ELSE
PRINT AT(45,23);"fichier non trouvé"
ENDIF
RETURN

```

Dans la procédure *sauvegarder*, j'ai renoncé à la condition EXIST, car on ne pourrait plus écraser un ancien fichier portant le même nom par un nouveau. Il suffit, pour qu'il y ait sauvegarde, que le fichier porte un nom (< > "").

Vous voyez que LINE INPUT peut servir pour divers périphériques d'entrée de données. Par la même occasion, nous en avons profité pour changer le nom des procédures. Il ne nous reste plus qu'à utiliser notre coffre-fort. Actualisons tout d'abord notre choix de procédures pour que nos deux procédures supplémentaires puissent être appelées :

```

PROCEDURE choix ! version-BOX
:
SELECT choix$
:
:
CASE "2","S"
sauvegarde
CASE "3","C"
charger
:
:
RETURN

```

ou encore :

```

PROCEDURE choix l version-GEM
:
SELECT choix%
:
:
CASE 12
sauvegarder
CASE 13
charger
:
:
RETURN

```

N'oubliez pas d'effacer éventuellement les procédures factices (*dummy*) portant le même nom ! Pour tester tout cela, entrez quelques données par la procédure *enregistrer*, affichez-les par la procédure *afficher*. Inventez alors un nom de fichier dont vous savez qu'il n'existe pas encore sur votre disquette, par exemple SAFE.DAT.

### 14.3. Sauvegarder et recharger

Choisissez maintenant la procédure *sauvegarde* : après avoir saisi le nom de fichier que vous avez inventé, l'unité de disquette devrait se mettre en marche et effectivement sauvegarder vos données. Terminez votre programme et relancez-le encore une fois : en lançant la procédure *afficher*, vous constatez que les données ne figurent effectivement plus dans les champs du tableau.

Voilà venu le moment décisif : vous sélectionnez le point du menu qui déclenche la procédure *charger*, après quoi vous entrez le nom du fichier (dont vous vous souvenez, j'espère). L'unité de disquette se met en marche..., et le programme GFA Basic s'interrompt brutalement en vous envoyant un message d'erreur signalant que vous êtes à la fin du fichier. Que se passe-t-il ?

Admettons que *max%* ait la valeur 9, signifiant qu'on peut saisir ou traiter au maximum 9 fiches. Admettons que, pour vos tests, vous vous soyez limité à la saisie de 3 fiches, que vous venez de sauvegarder sur la disquette : la variable numérique *no%* avait donc la valeur 3.

En relançant le programme, *no%* est ramené à 0. Si vous appelez maintenant la procédure de *charger* les trois fiches sont naturellement chargées depuis la disquette dans l'ordinateur. Examinez la condition provoquant la répétition de l'enregistrement des fiches :

```

:
WHILE no% < max%
  INC no%
  LINE INPUT #1, nom$(no%)
  INPUT #1, rue$(no%)
  INPUT #1, adresse$(no%)
  INPUT #1, telephone$(no%)
WEND
:

```

Ceci signifie que votre ordinateur tente de charger des fiches jusqu'à ce que *no%* ait atteint la valeur de *max%*. Si, comme dans notre exemple, vous n'avez saisi que 3 fiches, vous en avez donc sauvegardé sur la disquette moins que la valeur *max%*, et vous contraignez votre pauvre Atari à déclarer forfait ! Nous voilà bien obligé de l'aider !

Il faut ajouter une condition à la procédure *charger*, qui n'autorise le processus de chargement que s'il y a effectivement des données sur la disquette. La raison de l'interruption dans le programme était que le fichier prenait fin *trop tôt*. Il nous faut donc une condition tenant compte de la marque de fin du fichier. Vous savez que le GFA Basic attribue une marque de fin de fichier à un fichier sauvegardé sur une disquette. Cette marque s'appelle prosaïquement EOF (= End Of File).

L'ordinateur doit donc interrompre le processus de chargement au plus tard lorsqu'il rencontre cette marque. Autrement dit : le chargement des données doit se répéter tant qu'on n'a pas atteint la marque End Of File. Ainsi modifiée, notre procédure de chargement devrait tourner :

```

PROCEDURE charger
,
LOCATE 20,15
INPUT "nom du fichier :", nomfichier$

IF EXIST(nomfichier$)
  OPEN "I", #1, nomfichier$
  WHILE NOT EOF(#1) AND no% < max%
    INC no%
    LINE INPUT #1, nom$(no%)
    INPUT #1, rue$(no%)
    INPUT #1, adresse$(no%)
    INPUT #1, telephone$(no%)
  WEND
  CLOSE #1
ELSE
  PRINT AT(45,23); "fichier non trouvé"
  PAUSE 100
ENDIF
RETURN

```

Le (#1) derrière EOF désigne le numéro du canal. Il y a chargement jusqu'à ce que le coffre-fort soit entièrement vidé (=NOT EOF(#1)) et tant qu'il y a de la place chez nous (no% < max%). Pour que vous ayez le temps de lire un éventuel message, j'ai inséré la commande PAUSE : le nombre qui suit donne la durée de l'affichage en cinquantième de seconde, ce qui fait ici 2 secondes. Cela permet l'interruption momentanée du programme.

Et si vous ne l'avez pas encore fait tout seul, vous devez, dans la procédure *sauvegarder*, remplacer la boucle WHILE par FOR...NEXT, puisque le no% (= le nombre) est connu :

```
PROCEDURE sauvegarder
LOCAL aff%
'
LOCATE 23,15
INPUT "nom du fichier :",nomfichier$
IF nomfichier$ < > ""
OPEN "O",#1,nomfichier$
FOR aff% = 1 TO no%
PRINT #1,nom$(aff%)
PRINT #1,rue$(aff%)
PRINT #1,adresse$(aff%)
PRINT #1,telephone$(aff%)
NEXT aff%
CLOSE #1
ENDIF
RETURN
```

Vous pouvez être satisfait de votre banque, puisque vous pouvez maintenant y abriter des données en toute sécurité, et que vous pouvez les reprendre. On pourrait toutefois encore améliorer quelque chose : le message *fichier non trouvé* pourrait s'afficher sous une autre forme, en s'inscrivant dans une boîte d'alarme à l'aide de l'instruction

```
ALERT 1,"fichier non trouvé",1," OK |dommage",back%
```

J'en profite pour vous apprendre une nouvelle instruction :

```
FILESELECT "A : \*.DAT","SAFE.DAT",nomfichier$ ! chargement
```

ou

```
FILESELECT "A : \*.DAT","",nomfichier$ ! sauvegarde
```

Elle irait très bien avec la version GEM de notre *mini-banque de données*. La fenêtre de sélection de fichier irait bien aussi avec notre version *encadrée*. Grâce à l'instruction filesect qui met en place une boîte de sélection de fichier

```
FILESELECT <dossier>,<sélection_fichier>,nomfichier$
```

vous ouvrez une boîte contenant un répertoire partiel de votre disquette, de la même façon par exemple que lors du chargement ou de la sauvegarde du GFA Basic. Vous pouvez sélectionner un fichier à l'aide de la souris ou entrer un nom de fichier par le clavier. Le dossier actualisé se trouve sous *INDEX* au-dessus du répertoire partiel, le nom du fichier choisi se trouve sur la ligne *sélection*, où on peut entrer un nouveau nom. Comme vous le constatez pour la sauvegarde, vous ne pouvez entrer que "" pour *sélection\_fichier*.

## 14.4 Transfert de données

Nos procédures de sauvegarde et de chargement sont encore plus jolies, et elles s'écrivent :

```
PROCEDURE sauvegarder
LOCAL aff%
,
FILESELECT "A:\*.DAT","",nomfichier$
IF nomfichier$ < > ""
OPEN "O",#1,nomfichier$
FOR aff% = 1 TO no%
PRINT #1,nom$(aff%)
PRINT #1,rue$(aff%)
PRINT #1,adresse$(aff%)
PRINT #1,telephone$(aff%)
NEXT aff%

CLOSE #1
ENDIF
RETURN
,
PROCEDURE charger
,
FILESELECT "A:\*.DAT","SAFE.DAT",nomfichier$
IF EXIST(nomfichier$)
OPEN "I",#1,nomfichier$
WHILE NOT EOF(#1) AND no% < max%
INC no%
LINE INPUT #1, nom$(no%)
INPUT #1, rue$(no%)
INPUT #1, adresse$(no%)
INPUT #1, telephone$(no%)
WEND
CLOSE #1
ELSE
ALERT 1,"fichier non trouvé",1," OK |dommage",back%
```

```
ENDIF
RETURN
```

Si vous cliquez sur *annuler*, *nomfichier\$* est une chaîne vide (""). C'est pourquoi le processus de sauvegarde ne se déclenche pas. Lors du chargement, la procédure se termine dans ce cas par le message *fichier non trouvé*.

L'utilisation des procédures de sauvegarde et de chargement que nous venons d'écrire ne devrait pas se limiter à notre programme de mini-banque de données. Vous devriez pouvoir vous en servir pour l'archivage des images que vous avez créées en mode graphique et que vous avez mémorisées sous forme de chaîne grâce à GET (chapitre 9)

```
PROCEDURE pic_save(nomfichier$,image$)
OPEN "O",#1,nomfichier$
PRINT #1,image$
CLOSE #1
RETURN
```

Voilà votre image (= picture) en sécurité (= save) sur la disquette. Pour le chargement :

```
PROCEDURE pic_load(nomfichier$,VAR image$)
OPEN "I",#1,nomfichier$
INPUT #1,image$
CLOSE #1
RETURN
```

Par cette procédure et les instructions PUT appropriées, vous retrouvez votre image sur l'écran. Attention : *image\$* doit être ici un paramètre variable, car cette procédure doit redonner l'image chargée.

Il est intéressant de pouvoir transférer des données d'un fichier à un autre. Ceci correspond en principe au transfert des données du clavier (= appareil de saisie) vers l'écran (= appareil de sortie). Cette fois cependant, c'est l'unité de disquette qui fait office d'appareil de saisie et de sortie à la fois.

```
PROCEDURE copier(fichier.source$,fichier.cible$)
LOCAL text$
'
IF EXIST(fichier.source$)
OPEN "I",#1,fichier.source$
OPEN "O",#2,fichier.cible$
'
WHILE NOT EOF(#1)
LINE INPUT #1,text$
```

```

PRINT #2,text$
WEND
CLOSE
ELSE
  ALERT 1,"rien ne vient de rien!",1," OK|dommage",back%
ENDIF
RETURN

```

Nous devons ici utiliser les canaux 1 et 2 pour que les données ne se *tamponnent* pas lors de leur entrée-sortie. CLOSE sans plus de précision referme tous les fichiers ouverts. Vous pouvez insérer cette procédure dans votre programme de mini-banque de données, par exemple pour avoir un exemplaire *back-up* de vos fichiers.

La routine de recopiage peut d'ailleurs être utilisée aussi pour les fichiers *LST* sauvegardés par Save,A. Faites un essai :

```

REM recopiage
,
FILESELECT "a:\*.LST",,,,fichier.source$
FILESELECT "a:\*.BAK",,,,fichier.cible$
IF fichier.source$ < > "" AND fichier.cible$ < > ""
  copier(fichier.source$,fichier.cible$)
ELSE
  ALERT 2,"pas de copie sans nom !",2," OK |non",back%
ENDIF

END
,
PROCEDURE copier(fichier.source$,fichier.cible$)
  LOCAL text$
  ,
  IF EXIST(fichier.source$)
    OPEN "I",#1,fichier.source$
    OPEN "O",#2,fichier.cible$
    ,
    WHILE NOT EOF(#1)
      LINE INPUT #1,text$
      PRINT #2,text$
    WEND
    CLOSE
  ELSE
    ALERT 1,"rien ne vient de rien !",1," OK|dommage",back%
  ENDIF
  RETURN

```

Faites des essais, exercez-vous, y compris avec d'autres données comme par exemple des chiffres. Utilisez de préférence une disquette ne contenant pas de fichier important ! Il faut encore que vous connaissiez une autre possibilité



d'ouvrir un fichier. Dans la procédure *chargement*, vous pourriez remplacer la ligne

```
OPEN "O",#1,nomfichier$
```

par

```
OPEN "A",#1,nomfichier$
```

ou la faire figurer comme variante possible. "A" est l'abréviation de APPEND : s'il existe un fichier de même nom lors de la sauvegarde des données, il ne sera pas écrasé, les données nouvelles viendront s'ajouter aux données préexistantes. S'il n'existe pas encore de fichier portant le nom choisi, il sera créé. Ce serait peut-être bien pour votre mini-banque de données ?

## 14.5. Résumé

Vous connaissez maintenant l'essentiel (mais pas tout, et de loin !) sur la manipulation des fichiers par rapport à l'unité de disquette. En tout cas, vos données peuvent être mises en sécurité sur une disquette et vous pouvez les y rechercher à tout moment.

Pour votre confort, vous connaissez l'instruction

```
FILESELECT <dossier> , <sélection_fichier> , nomfichier$
```

qui sert à ouvrir une boîte de sélection des fichiers : on peut préciser auparavant le dossier ainsi que le fichier choisis. La variable *nomfichier\$* contient le nom du fichier à charger ou à sauvegarder. Si vous sélectionnez *annuler*, *nomfichier\$* prend la valeur d'une chaîne vide.

Vous pouvez contrôler l'existence d'un fichier sur la disquette ou le disque dur grâce à *EXIST(nomfichier\$)* :

```
EXIST(nomfichier$) = TRUE      (< > 0) le fichier existe
EXIST(nomfichier$) = FALSE    (= 0) le fichier n'existe pas
```

Vous savez qu'un canal mène à l'unité de disquette ainsi qu'à d'autres appareils périphériques, et qu'il est désigné par un numéro.

```
OPEN "I",#no_canal,nomfichier$ ("I" = INPUT)
```

vous permet d'ouvrir un fichier pour le lire. Un pointeur est placé au début du fichier. Si le fichier n'existe pas, vous recevez un message d'erreur.

```
OPEN "O",#no_canal,nomfichier$ ("O" = OUTPUT)
```

vous permet d'ouvrir un fichier pour y écrire. Un pointeur est placé au début du fichier. Si le fichier existe déjà, son contenu sera écrasé et remplacé par les nouvelles données.

```
OPEN "A",#no_canal,nomfichier$ ("A" = APPEND)
```

vous permet d'ouvrir un fichier pour y écrire. Un pointeur est placé à la fin du fichier. Si le fichier existe déjà, les nouvelles données sont ajoutées aux anciennes. Vous vous souvenez que vous pouvez, grâce à *EOF(#canal\_no)*, contrôler si vous êtes bien arrivé à la fin d'un fichier.

Après avoir ouvert le fichier, vous pouvez lire les données et les écrire sur une disquette par les instructions :

```
INPUT #no_canal,<variable>
```

ou

```
LINE INPUT #no_canal,<variable>
```

et

```
PRINT #no_canal,<variable>
```

N'oubliez pas de refermer un fichier ouvert par "CLOSE#no\_canal" ou, si vous en avez ouvert plusieurs, par CLOSE écrit seul.

Vous savez que vous pouvez *entendre* ce que vous avez écrit grâce à *PRINT CHR\$(7)*, et enfin que vous pouvez faire une *PAUSE <durée>* ...

## Chapitre 15

### Traitement des données

Après avoir saisi quelques données, les avoir contemplées, les avoir transportées aller-retour de la RAM à la disquette, vous avez senti monter le désir de disposer d'une procédure de correction. Car vous avez vraisemblablement fait des fautes de frappe, et vous vous êtes de plus en plus énervé de ne pouvoir les corriger. D'ailleurs, vous n'avez sans doute pas encore calmé votre soif de disposer de possibilités plus étendues pour votre mini-banque de données ou pour d'autres projets.

#### 15.1. Il faut pouvoir apporter des corrections...

Essayons tout d'abord de satisfaire votre désir de rendre le traitement de vos données plus confortable. Peut-être serez vous déjà bien content de pouvoir modifier la procédure de saisie de telle sorte qu'elle permette la répétition et donc la correction d'une saisie quelconque. Il faut cependant pour cela insérer certains signes signalant à l'ordinateur qu'une donnée va être corrigée.

Il serait intéressant d'inclure la procédure de sortie dans cette modification, car la correction de données n'est finalement rien d'autre qu'une combinaison d'entrées et de sorties. Pour fabriquer une procédure de correction autonome, nous allons réutiliser des segments de programme déjà écrits, ce qui va nous donner notre première version *brute* en mélangeant le texte de la procédure de sortie et de saisie :

```
PROCEDURE correction  
  LOCAL clavier$,aff%  
,  
  masque
```

```

col&=25
WHILE clavier$ < > CHR$(27) AND aff% < no%
  INC aff%
  tete(aff%,"correction")
  formulaire(7,5)
  '
  PRINT AT(col&,7);nom$(aff%)  ! affichage
  PRINT AT(col&,11);rue$(aff%)
  PRINT AT(col&,15);adresse$(aff%)
  PRINT AT(col&,19);telephone$(aff%)
  '
  LOCATE 7,col&                ! saisie
  LINE INPUT "",nom$(aff%)
  LOCATE 11,col&
  INPUT "",rue$(aff%)
  LOCATE 15,col&
  INPUT "",adresse$(aff%)
  LOCATE 19,col&
  INPUT "",telephone$(aff%)
  '
  bas(clavier$)
WEND
RETURN
    
```

Vous voyez s'afficher d'abord la fiche complètement écrite, qu'il vous faut ensuite réécrire totalement. Vous voyez tout de suite la faiblesse de notre procédure : pour corriger l'une ou l'autre donnée, il faut re-saisir toute la fiche, et donc toutes les données entrées auparavant correctement !

Si vous voulez faire tourner cette version, coupez-la dans la routine de sélection avec *aide(sos)*. Cette procédure sera alors appelée en entrant "5" ou "O" ou menu no 23.

Au lieu de lancer la routine "Aider" vous exécuterez plutôt une procédure "correction".

```

PROCEDURE choix ! version-BOX
:
SELECT choix$
:
:
CASE "5","O"
  correction
:
:
RETURN
    
```

ou encore :

```
PROCEDURE choix ! version-GEM
```

```
  :  
  SELECT choix%
```

```
  :  
  CASE 23  
  correction
```

```
  :  
  RETURN
```

Comme nous aimons les changements, nous pourrions insérer devant chaque instruction INPUT un contrôle nous demandant si nous voulons modifier la valeur des données ou les laisser inchangées. Ceci serait réalisable grâce à INKEY\$ : il suffirait par exemple d'actionner la touche <Return> pour valider la valeur existante, et d'appuyer "#" (ou n'importe quelle autre touche que vous choisirez) pour passer en correction :

```
PROCEDURE controle(colonne&,ligne&,actuel%,VAR correcte$())
```

```
LOCAL clavier$
```

```
  LOCATE ligne&,colonne&
```

```
  REPEAT
```

```
    clavier$ = INKEY$
```

```
  UNTIL clavier$ = "#" OR clavier$ = CHR$(13)  !return
```

```
  IF clavier$ = "#"
```

```
    PRINT SPACES$(74-colonne&);  !effacer la saisie
```

```
    LOCATE ligne&,colonne&
```

```
    LINE INPUT correcte$(actuel%)
```

```
  ENDIF
```

```
RETURN
```

L'ordinateur interroge le clavier jusqu'à ce qu'une des touches <Return> ou "#" (code ASCII = 13) soit actionnée. Dans ce dernier cas, les données saisies précédemment sont effacées de la ligne concernée, ce qui élimine une des faiblesses de notre procédure de correction. Il faut donc indiquer en même-temps la position actualisée du curseur. En actionnant <Return> par contre, les données saisies sont validées.

"VAR correcte\$()" assure le transfert indirect d'un tableau entier, ce qu'on constate par les parenthèses vides placées à la fin de l'expression. Indirect, car la procédure n'apprend que l'endroit où se trouve le tableau dans les mémoires de travail, et elle doit rechercher elle-même la chaîne concernée à l'aide du paramètre *actuel%*. La définition VAR garantit que la correction éventuelle parvienne bien au bon endroit.

Il ne nous reste plus qu'à modifier en conséquence notre procédure de correction :

```
PROCEDURE correction
LOCAL clavier$,aff%
,
masque

col& = 25
WHILE clavier$ < > CHR$(27) AND aff% < no%
  INC aff%
  tete(aff%,"correction")
  formulaire(7,5)
  ,
  ' affichage ou correction au choix :
  ,
  PRINT AT(col&,7);nom$(aff%)
  LOCATE col&,7
  controle(col&,7,aff%,nom$())
  PRINT AT(col&,11);rue$(aff%)
  LOCATE col&,11
  controle(col&,11,aff%,rue$())
  PRINT AT(col&,15);adresse$(aff%)
  LOCATE col&,15
  controle(col&,15,aff%,adresse$())
  PRINT AT(col&,19);téléphone$(aff%)
  LOCATE col&,19
  controle(col&,19,aff%,téléphone$())
  ,
  bas(clavier$)
WEND
RETURN
```

Vous constatez qu'ici aussi, l'ensemble du tableau d'une variable-chaîne est pris comme paramètre à chaque fois qu'on appelle la procédure *controle* : vous devez adjoindre des parenthèses vides à chaque nom concerné.

J'ai *mixé* les procédures de correction et d'affichage de telle sorte que les données contenues sur une fiche ne soient plus affichées en même temps mais l'une après l'autre. Vous pouvez modifier cela si bon vous semble.

## 15.2. .... et aussi des améliorations

Nous devrions apporter encore une amélioration à ces procédures de correction, d'affichage et de saisie pour pouvoir feuilleter notre fichier non seulement vers l'avant mais aussi vers l'arrière. Cela ne nous coûtera pas

beaucoup de travail puisque nous allons réutiliser une procédure déjà écrite en la modifiant un peu :

```
PROCEDURE bas(VAR fiche%,clavier$)
PRINT AT(4,23);" < Esc >  FIN";SPACE$(10);" < espace >  retour";
PRINT SPACE$(22);" < sinon >  continuer";
REPEAT
  clavier$ = INKEY$
UNTIL clavier$ < > ""
IF clavier$ = " "
  DEC fiche%
  IF fiche% > 0
    DEC fiche%
  ENDIF
ENDIF
RETURN
```

Cette procédure comporte deux paramètres variables : vous devez contrôler si ces deux variables sont bien transmises aux trois procédures concernées (enregistrer, afficher, correction) car la routine de la ligne du bas va indiquer la direction dans laquelle vous feuilletez les fiches :

```
bas(no%,clavier$) ! enregistrer
bas(aff%,clavier$) ! afficher
```

Le comptage en descendant -décrémenter- se produit deux fois pour chaque fiche, car la valeur du paramètre VAR *fiche%* est redonnée à la procédure appelée qui la réaugmente aussitôt de 1 unité sous *no%* ou *aff%*. On ne peut naturellement *décrémenter* qu'une seule fois la fiche du début, puisqu'aucune fiche ne la précède.

Si cette procédure de correction vous plaît et que vous voulez définitivement la reprendre dans votre programme de mini-banque de données, vous devriez vous servir d'une procédure intermédiaire vous permettant de choisir entre la saisie, l'affichage et la correction :

```
PROCEDURE traitement
LOCAL travail$
' choix
col& = 25
CLS
LOCATE col&,7
PRINT " TRAITEMENT"
LOCATE col&,10
PRINT "1.Entree des donnees      E"
LOCATE col&,12
PRINT "2.Affichage des donnees   A"
```

```
LOCATE col&,14
PRINT "3.Correction des donnees    C"
LOCATE col&,17
PRINT "0. FIN                    < Esc > "
'
'sélection
REPEAT
  travail$ = UPPER$(INKEY$)
UNTIL travail$ < > ""
SELECT travail$
CASE "0",27    !Esc
  enregistrer
CASE "1","E"
  afficher
CASE "2","A"
  correction
DEFAULT
  traitement
ENDSELECT
CLS
RETURN
```

Vous pouvez maintenant au choix : entrer de nouvelles données, afficher des données existantes, les modifier si nécessaire immédiatement. Ceci libère l'option 4 de votre menu principal 4. *Affichage/Impression* qui pourrait servir par exemple à la sortie sur une autre imprimante ou vers un autre ordinateur via un modem. Je vous laisse le soin de confectionner des cadres pour ce menu, pour lequel vous pouvez reprendre la version GEM de notre deuxième version du menu principal.

Vous voulez peut-être remplacer totalement les procédures de sortie et entrée par la procédure de correction ? ceci vous épargnerait le sous-menu de traitement. Il n'y a qu'une seule petite modification à apporter, car les procédures d'entrée et de sortie n'ont pas les mêmes variables pour compter les fiches

*no%* compte les fiches qui viennent d'être remplies : cette variable sert donc quasiment de pointeur vers la dernière fiche écrite. Les autres fiches allant jusque *max%* sont encore vides.

*aff%* par contre compte les fiches qui sont affichées ou transmises pour être traitées. On pourrait en fait tout de suite supprimer la procédure d'affichage, car dans la pratique il est rare qu'on ne veuille qu'afficher des données sans se donner la possibilité de les corriger immédiatement s'il en est besoin.

S'il n'y a pas de données, les procédures de correction et d'affichage ne peuvent pas être mises en oeuvre, car la condition *aff% < no%* n'est pas remplie. Il vous



suffit de modifier cette condition dans les procédures d'affichage et de correction, de la remplacer par *aff% < max%* et vous pourrez feuilleter en avant ou en arrière toutes vos fiches, qu'elles soient remplies ou encore vides.

La procédure de correction vous permet alors d'entrer des données sur n'importe quelle fiche, ce qui rend superflue la routine de saisie. En supprimant le pointeur *no%*, vous courez le risque de ne plus maîtriser l'ensemble des données déjà saisies ; de plus vous transmettez même les fiches vides lors des sauvegardes, des sorties sur papier ou des transmissions par modem !

Nous devons donc conserver la variable-pointeur *no%* qui doit compter les nouvelles fiches : sitôt que la valeur de *aff%* dépasse celle de *no%*, ce dernier doit augmenter. En intégrant tout cela dans notre routine de correction, nous devrions pouvoir nous passer de la procédure d'enregistrement :

```
PROCEDURE correction
LOCAL clavier$,aff%
,
masque
col& = 25
WHILE clavier$ < > CHR$(27) AND aff% < max%
  INC aff%
  IF aff% > no%
    INC no%
  ENDIF
  tete(aff%,"correction")
  formulaire(7,5)
  LOCATE 23,4
  PRINT "<#> correction";SPACE$<39>;"<Return> entree OK"
,
'affichage ou correction au choix :
,
PRINT AT(col&,7);nom$(aff%)
LOCATE col&,7
controle(col&,7,aff%,nom$())
PRINT AT(col&,11);rue$(aff%)
LOCATE col&,11
controle(col&,11,aff%,rue$())
PRINT AT(col&,15);adresse$(aff%)
LOCATE col&,15
controle(col&,15,aff%,adresse$())
PRINT AT(col&,19);telephone$(aff%)
LOCATE col&,19
controle(col&,19,aff%,telephone$())
,
bas(clavier$)
WEND
RETURN
```

Ne détruisez quand même pas les procédures d'entrée et d'affichage, car il n'est jamais inutile de pouvoir choisir !

### 15.3. Classement des données

Je vais maintenant vous faire découvrir quelques utilitaires dont dispose le GFA Basic surtout dans la version 3.0. Vous pouvez par exemple trier vos données selon deux méthodes différentes, grâce aux instructions QSORT et SSORT. Ces instructions permettent de trier les éléments d'un tableau d'après leur taille. "Q" signifie ici *quick* c'est à dire *rapide*, et il s'agit du tri le plus rapide qui suffit dans la plupart des cas. La méthode Shellsort (d'où le "S") s'emploie dans des cas particuliers. Essayez ce programme :

```
REM Classement
DIM nom$(10)
CLS
FOR i%=0 TO 10
  INPUT nom$(i%)
NEXT i%
CLS
FOR i%=0 TO 10
  PRINT nom$(i%)
NEXT i%
PRINT
QSORT nom$()  ! ou SSORT
FOR i%=0 TO 10
  PRINT nom$(i%)
NEXT i%
```

Les noms sont classés en un clin d'oeil. Ce petit programme de test ne vous permettra pas de constater une grande différence au niveau de la rapidité du tri entre QSORT et SSORT, car le nombre des données à trier est trop faible.

Si vous voulez tester la différence entre les deux méthodes de tri sur de grandes masses de données, vous pouvez remplir un tableau d'une foule de nombres aléatoires, pour en demander ensuite le tri à chacune des procédures.

QSORT et SSORT classent les chaînes dans l'ordre du tableau des codes ASCII (voir Annexe C). Ceci signifie par exemple que les lettres affectées d'un tréma ou les caractères spéciaux se retrouvent à la fin de l'alphabet, ce qui ne correspond pas au classement normal. Nous pouvons modifier cela dans le GFA Basic 3.0 :

```

REM Classement en tenant compte des trémas
4
DIM trema| (256)
FOR n| = 0 TO 255
  trema|(n|) = n| ! code Ascii pour le caractère normal
NEXT n|

FOR u| = 1 TO 7
  READ um$, norm$
  code| = ASC(um$)
  trema|(code|) = ASC(norm$) !code Ascii modifié
NEXT u|

DATA Ä, A, Ö, O, Ü, U, ä, a, ö, o, u

DIM nom$(10)
CLS
FOR i% = 0 TO 10
  INPUT nom$(i%)
NEXT i%

CLS
FOR p% = 0 TO 10
  PRINT nom$(p%)
NEXT p%
PRINT
QSORT nom$() WITH trema| ()
FOR p% = 0 TO 10
  PRINT nom$(p%)
NEXT p%

```

WITH permet d'introduire un tableau ne respectant pas la norme ASCII, et qui devient le tableau de référence pour le classement. Le tréma a été ici défini comme BYTE (|). Vous pouvez vous-même déterminer la *direction* du classement, du plus petit au plus grand ou inversement :

```

QSORT <tableau> (+) ou SSORT <tableau> (+) tri ascendant
QSORT <tableau> (-) ou SSORT <tableau> (-) tri descendant

```

Si les parenthèses restent vides, le tri sera toujours ascendant. Puisque nous en sommes au tri et au classement, je vais vous indiquer quelques commandes supplémentaires offertes par le GFA Basic 3.0 :

L'instruction "INSERT nom\$(no%) = nouveau-nom\$" ajoute un nouvel élément dans un tableau de chaînes de caractères ou de chiffres à l'endroit désigné par *no%*. Tous les éléments suivants sont décalés d'une place vers l'avant (+1). Mais ceci n'aggrandit pas le tableau, ce qui peut provoquer la disparition du dernier élément !

L'instruction *DELETE nom\$(no%)* efface l'élément qui se trouve à la place *no%*. Tous les éléments suivants sont décalés d'une place vers l'arrière (1). Le dernier élément prend la forme d'une chaîne vide ("").

L'instruction *SWAP <variable\_1>, <variable\_2>*, permute deux variables de même type. Elle peut être aussi utiliser pour changer deux tableaux. Cet échange se fera très rapidement, puisque seuls les descripteurs correspondants sont en réalité échangés. Avec toute l'expérience que vous avez accumulée et la joie que vous montrez pour l'expérimentation, vous êtes sûrement capable maintenant d'allonger votre programme de mini-banque de données en y insérant les procédures de tri et de recherche.

## 15.4. Toutes sortes de choses bien utiles

Le GFA Basic vous offre aussi des facilités pour la gestion de vos fichiers : DIR et FILES servent à afficher des répertoires du contenu des disquettes ou du disque dur. DIR ne vous donne que la liste des noms des fichiers sur un lecteur déterminé, alors que FILES y ajoute leur longueur, leur date-système et l'heure-système.

Dans les deux cas, vous pouvez ajouter le nom d'un répertoire ou un masque (par exemple *\*.LST*); en adjoignant encore TO, vous pourrez transférer des répertoires entiers ou des fichiers de même extension vers une disquette ou un disque dur plutôt que vers l'écran.

`DIR "A:\*.*)" TO "LST:" (ou "PRN:")`

ou encore

`FILES "A:\*.*)" TO "LST:" (ou "PRN:")`

vous permet de sortir sur l'imprimante le répertoire de votre disquette. Si vous voulez le répertoire des programmes du GFA Basic, remplacez le masque *"\*.\*)"* par *"\*.GFA", "\*.BAS"* ou *"\*.LST"*.

Si vous voulez effacer un fichier lorsque vous êtes dans un programme (comme par exemple celui de votre mini-banque de données), vous utilisez *KILL <nom du fichier>*. Si seul le nom du fichier vous déplaît, changez-le par *RENAME <ancien nom> AS <nouveau nom>*. Vous pouvez aussi utiliser NAME, mais attention à l'utilisation du mot *name* dans l'écriture des variables ou des procédures !

Ces deux instructions devraient d'ailleurs nous donner un peu plus de confort dans notre procédure de sauvegarde :

```
PROCEDURE sauvegarder
LOCAL aff%
'
FILESELECT "A:\*.DAT",nomfichier$
IF nomfichier$ < > ""
  IF EXIST(nomfichier$)
    backup$ = nomfichier$
    MID$(backup$,LEN(backup$)-2) = "BUP"
    IF EXIST(backup$)
      KILL backup$
    ENDIF
    RENAME nomfichier$ AS backup$ ! ou NAME
  ENDIF
OPEN "O",#1,nomfichier$
FOR aff% = 1 TO no%
  PRINT #1,nom$(aff%)
  PRINT #1,rue$(aff%)
  PRINT #1,adresse$(aff%)
  PRINT #1,telephone$(aff%)
NEXT aff%
CLOSE #1
ENDIF
RETURN
```

Ce processus ne se déclenchera naturellement que s'il existe un fichier de même nom :

```
IF EXIST(nomfichier$)
```

Le nom du fichier sous lequel vos données doivent être abritées est d'abord pourvu d'un suffixe, d'une *extension*. Pour cela, il faut connaître la longueur de ce nom (LEN). MID\$ va écraser l'ancienne extension (par exemple DAT) et la remplacer par la nouvelle. Si BUP ne vous plaît pas, prenez donc BAK, mais pensez au fait que la version 3.0 du GFA Basic utilise aussi cette extension ! Le changement de nom ne doit pas concerner *nomfichier\$*, mais doit être attribué à une autre variable :

```
backup$ = nomfichier$
MID$(backup$,LEN(backup$)-2) = "BUP"
```

Mais il pourrait arriver qu'il y ait déjà un autre Backup, qui devra être effacé :

```
IF EXIST(backup$)
  KILL backup$
ENDIF
```

Le nom du fichier peut enfin devenir celui de son Backup :

```
RENAME nomfichier$ AS backup$
```

Vous pourriez aussi enrichir vos programmes ultérieurs de deux autres instructions du GFA Basic. Essayez cette *autre manière* :

```
REM embranchements
EVERY 20 GOSUB singsang
'

CLS
FOR i& = 1 TO 420
  hasard& = INT(RND*94) + 33
  PRINT CHR$(hasard&); " ";
NEXT i&
'

AFTER 400 GOSUB message
RBOX 70,360,600,390
PRINT AT (12,24);"pour terminer,appuyez sur une touche !
      vous avez deux secondes !";

REPEAT
UNTIL INKEY$ < > "" OR back% = 2  ! oh non !
SOUND 1,0,0,0,0
'

PROCEDURE singsang
  ton& = INT(RND*12) + 1
  SOUND 1,15,ton&,4,2
RETURN
'

PROCEDURE message
  CLS
  ALERT 3,"trop tard ! | le carroussel repart !",
    1,"ah bon | oh non !",back%
RETURN
```

Ne vous attendez pas à ce que je fournisse des explications ! Vous avez peut-être compris que la variable *back%* est bien utile : elle est utilisée ici dans la condition *back% = 2* pour contrôler si vous avez cliqué sur le bouton no 2 portant l'indication "oh non !". Je vous recommande tout spécialement ces deux instructions :

```
EVERY <rythme> GOSUB <procédure>
AFTER <rythme> GOSUB <procédure>
```

La première provoque la répétition perpétuelle d'une procédure après un certain délai (en 200tièmes de secondes), la deuxième répète une fois la procédure après un certain délai (en 200tièmes de secondes également).

Ces deux instructions peuvent être utilisées en liaison par exemple avec des changements d'heure ou de date, pour lesquels vous avez de plus les fonctions DATE\$, TIME\$ et SETTIME\$.

## 15.5. Vous allez savoir marcher tout seul ?

Après tout ce que j'ai fait pour vous, je peux maintenant vous laisser élaborer le *siège* de votre banque de données à titre d'exercice grâce aux nouveautés que je viens de vous présenter.

Désirez-vous maintenant protéger les efforts que vous avez produits pour faire cette banque ? Avec

PSAVE <nom du programme>

L'instruction PSAVE est employée pour que le fichier spécifié soit stocké avec une protection (anti-list ne pourra plus être listé)

Vous donnez à vos programmes une protection contre la visualisation, ce qui vous empêchera (vous et toute autre personne) de lister le texte du programme. Il est conseillé d'avoir auparavant sauvegardé un listing normal du programme ! Qui plus est, il suffira de LOAD pour le lancer à chaque chargement.

A un moment ou à un autre, vous aurez le sentiment qu'il n'y a plus rien à perfectionner dans votre programme, qu'il est parfait, et qu'il est temps de *l'émanciper* en coupant le cordon ombilical avec l'environnement GFA.

Cela n'est pas simple, car vous savez qu'aucun programme en GFA Basic ne pouvait jusqu'à présent tourner sans son interpréteur. C'est pourquoi vous avez de plus le fichier GFABASRO.PRG, ce qui signifie GFA-BASIC-Run-Only-Interpreter. C'est le même logiciel que celui que vous avez utilisé jusqu'à présent, il manque juste l'éditeur.

Cliquez deux fois sur GFABASRO.PRG : le fichier est chargé dans l'Atari ST, et une boîte de sélection apparaît. Si vous sélectionnez un programme en GFA Basic, il sera exécuté juste après son chargement. Cela ne fonctionnera qu'avec des programmes sauvegardés par SAVE ou PSAVE ; vous ne pourrez en aucun cas lancer ainsi des fichiers LST !

Encore autre chose : depuis le bureau GEM, cliquez une seule fois sur GFABASRO.PRG avec la souris pour provoquer son inversion-vidéo. Ouvrez

alors le menu des options et sélectionnez *installer une application*. Une boîte de dialogue apparaît dans laquelle figure déjà le nom GFABASRO.PRГ :

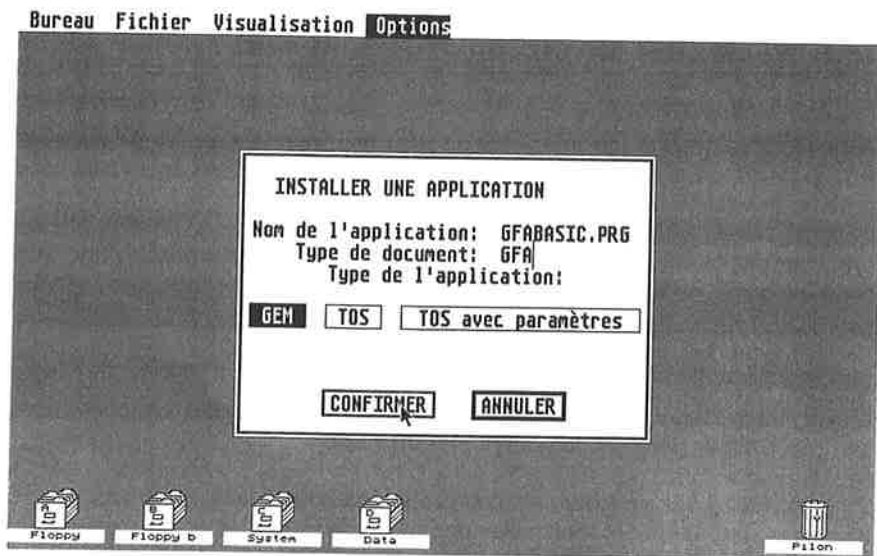


Figure 8 : interpréteur Run-Only

Entrez alors *GFA* ou *BAS* sous *type de document* (pas *LST*) et cliquez sur *confirmer*. A partir de ce moment, il vous suffira de cliquer sur un fichier *GFA* pour lancer aussitôt l'interpréteur Run-Only, qui chargera lui-même le programme *GFA* sélectionné et le fera tourner. Si vous sauvez ce choix en cliquant ensuite sur *sauvegarder le bureau*, votre installation de l'application s'inscrit dans le fichier *DESKTOP.INF* du *GEM*, et restera valable même après avoir éteint puis rallumé votre Atari. Attention, cela ne vaut que pour la disquette sur laquelle se trouve *GFABASRO.PRГ* avec le *DESKTOP.INF* adéquat !



## 15.6. Résumé

Vous avez amassé une multitude de connaissances qui vous seront utiles non-seulement pour votre programme de mini-banque de données mais aussi dans tous vos projets ultérieurs si vous disposez de la version 3.0, car la version 2.0 ne possédait que très peu des mots présentés ici (voir chapitre 16.).

En écrivant la procédure de correction, vous avez appris que *VAR <nom du tableau()>* permet d'attribuer indirectement comme paramètres des tableaux entiers de données à des procédures chargées du traitement. Vous pouvez classer rapidement vos données grâce à *QSORT <nom du tableau()>* ou *SSORT <nom du tableau()>*. Vous pouvez changer de critère de tri dans les tableaux de chaînes de caractères grâce à *WITH <tableau numérique>()*.

Vous pouvez disposer de la possibilité d'insérer ou d'effacer des éléments particuliers par *INSERT <nom du tableau()> = <nouvel élément>* et *DELETE <nom du tableau()>*. Vous pouvez permuter deux variables de même type (par exemple des éléments d'un tableau) en utilisant *"SWAP <variable\_1>, <variable\_2>"*.

Vous pouvez visualiser le contenu d'une disquette par *DIR* ou *FILES*, ce qui vous permet aussi de détourner la sortie par *TO* vers un fichier ou par exemple l'imprimante. Par ailleurs, *RENAME* ou *NAME <ancien nom> AS <nouveau nom>* vous permet de renommer un fichier et *KILL <nomfichier>* de l'effacer.

Pour que votre Atari ST fasse de temps en temps autre chose qu'exécuter les programmes ordinaires, pensez à *"EVERY <rythme> GOSUB <procédure>"*. Si vous voulez déclencher quelque chose au bout d'un certain laps de temps, utilisez *"AFTER <durée> GOSUB <procédure>"*.

N'hésitez pas enfin à protéger vos petits secrets grâce à *PSAVE <nom du programme>*. Si vous renoncez à l'éditeur, vous pouvez faire tourner vos programmes avec l'interpréteur Run-Only, et même automatiquement si vous le désirez.



## **Partie 4**

### **Au secours !**



## Chapitre 16

### Vue d'ensemble sur le GFA Basic

C'est vous que je viens d'entendre crier ainsi ? Que se passe-t-il ? Vous ne trouvez plus un des mots du GFA Basic ? Il y en a un autre que vous ne reconnaissez plus ? Une erreur est apparue ? Vous êtes à bout de nerfs ?

Dans ce dernier cas, consultez quelqu'un d'autre ! ou sinon commencez par faire une pause. Feuilletez ensuite lentement ce chapitre : vous y trouverez peut-être un des mots recherchés ou au moins une piste qui vous dépannera.

J'ai réuni ici tous les termes du GFA Basic que vous avez rencontrés tout au long de ce livre. Si vous voulez utiliser la forme abrégée d'un des termes GFA (qui sera alors complété par l'éditeur), elle est indiquée juste devant la forme développée. Les différences entre la version 2.0 et 3.0 sont signalées par des crochets.

Chaque terme est suivi d'une ligne comportant une brève explication ainsi que de la référence du chapitre dans lequel le terme concerné est expliqué plus complètement. Les mots suivis de trois points d'exclamation ont été modifiés par rapport à la version 2.0 ou n'y existe pas.

Vous trouvez ensuite une page concernant l'adaptation en GFA Basic 3.0 des programmes écrits en version 2.0.

## 16.1. Le vocabulaire

*ad*

*ADD variable, quantité*

Augmenter une variable numérique de la quantité indiquée (chap. 6)

*af !!!*

*AFTER temps GOSUB procédure version 3.0*

Appeler une procédure après un temps déterminé (chap. 15)

*a*

*ALERT symbole, texte, bouton return, bouton texte, back*

Fabriquer des messages d'alarme (fenêtre de dialogue) (chap. 10)

*Valeur AND Valeur*

Opération logique ET sur deux conditions (chap. 6)

*arr*

*ARRAYFILL nom du tableau(), nombre*

Remplir les champs numériques du tableau avec des valeurs (chap. 6)

*AS*

voir NAME | RENAME

*ASC(numéro)*

Retrouver le code ASCII/ST d'un caractère (chap. 6)

*BASE*

voir OPTION BASE

*bo*

*BOX x\_gauche, y\_haut, x\_droite, y\_bas*

trace un rectangle à partir des coordonnées x1,y1 et x2,y2 des coins opposés (chap. 9)

*ca !!!*

*CASE version 3.0*

voir SELECT

*CHR\$(nombre)*

Retrouver le caractère à partir du code ASCII/ST (chap. 6)

*CIRCLE rayon x,y*

Dessiner des cercles de rayon R dont le centre est défini par les coordonnées x,y (chap. 9)

*cle*

*CLEAR*

Effacer la mémoire des variables et des tableaux (chap. 6)

*cl*

*CLOSE [#Numéro de fichier]*

Fermer un fichier (chap. 14)

*cl w*

*CLOSEW Numéro de fenêtre*

Fermer une fenêtre GEM n (chap. 11)

*CLS*

Effacer le contenu de l'écran (chap. 10)

*co*

*COLOR Valeur de la couleur*

Déterminer la couleur utilisée (chap. 9)

*da*

*DATA Valeur [,...]*

Permet de stocker des valeurs qui pourront être lues par READ (chap. 6)

*DATE\$*

Indiquer la date-système [et la mettre à jour (3.0)] (chap. 13)

*DEC variable*

Diminuer (décrémenter) une variable de 1 unité (chap. 6)

*defa !!!*

*DEFAULT version 3.0*

voir SELECT

*defbi !!!*

*DEFBIT domaine d'application version 3.0*

Déterminer le champ d'application d'une variable booléenne (TRUE|FALSE) (chap. 6)

*defb !!!*

*DEFBYT domaine d'application version 3.0*

Déterminer le champ d'application d'une variable en tant que nombre entier (0 - 255) (chap. 6)

*deff*

*DEFFILL couleur, style, remplissage*

Déterminer le motif de remplissage (chap. 9)

*deffl !!!*

*DEFFLT domaine d'application*

Déterminer le champ d'application d'une variable en tant que nombre décimal (chap. 6)

*DEFN Nom[(argument)]*

Définir une fonction d'une seule ligne (chap. 8)

*defi !!!*

*DEFINT domaine d'application version 3.0*

Déterminer le champ d'application d'une variable en tant que nombre entier (+ | - 2 mrd) (chap. 6)

*de*

*DEFLINE style, épaisseur, forme au début, forme à la fin*

Déterminer la forme d'une ligne (chap. 9)

*deflis*

*DEFLIST Numéro*

Format du listing de programme (seulement en mode direct) (chap. 1)

*defmo*

*DEFMOUSE symbole*

Déterminer le symbole de la souris (chap. 9)

*defmo*

*DEFMOUSE string de forme*

Déterminer soi-même la forme du curseur de la souris (chap. 9)

*defs !!!*

*DEFSTR domaine d'application version 3.0*

Déterminer le domaine d'application d'une variable en tant que string (chap. 6)



*deft*

*DEFTEXT couleur,type,angle de l'affichage, grosseur*

Déterminer les attributs d'un texte graphique (chap. 9)

*defw !!!*

*DEFWRD domaine d'application version 3.0*

Déterminer le champ d'application d'une variable en tant que nombre entier (+ | - 32767) (chap. 6)

*Del*

*DELETE tableau (indice) version 3.0*

Détruire un élément d'un tableau (chap. 15)

*DIM Nom du tableau(numéro d'index [...]) [...]*

Définir et enregistrer les tableaux (chap. 6)

*DIM ? (Nom du tableau())*

Obtenir le nombre des éléments du tableau (chap. 12)

*DIR [masque] [TO fichier\imprimante]*

Afficher le répertoire du disque ou de la disquette (seulement les noms) (chap. 15)

*DIV variable, quantité*

Diviser une variable numérique par la quantité indiquée (chap. 5)

*Nombre DIV Nombre*

Division pour obtenir un nombre entier (chap. 6)

*DO*

*instruction(s)*

*LOOP*

Boucle sans condition de test C'est une boucle sans fin (chap. 5)

*!w|!u !!!*

*DO*

*instruction(s)*

*LOOP WHILE|UNTIL condition*

Boucle de répétition avec test de la condition à la fin (chap. 4)

*do w|do u !!!*

*DO WHILE|UNTIL condition*

*instruction(s)*

*LOOP*

Boucle de répétition avec test de la condition au début (chap. 4)

*DOWNTO*

voir FOR

*dr*

*DRAW x1,y1 (TO x2,y2 ...)*

Dessiner des points, des lignes, des polygones (chap. 9)

*dr !!!*

*DRAW string-tortue*

Graphique avec la tortue après des ordres brefs (chap. 9)

*ell*

*ELLIPSE x,y,x rayon,y rayon*

Dessiner des ellipses (chap. 9)

*el*

*ELSE*

voir IF

*el !!!*

*ELSE IF condition version 3.0*

voir IF

*END*

Terminer le programme (chap. 2)

*en*

*ENDIF*

voir IF

*endf !!!*

*ENDFUNC version 3.0*

voir FUNCTION

*ends !!!*

*ENDSELECT version 3.0*

voir SELECT

*EOF*(#numéro de fichier)

Marquage de fin du fichier (chap. 14)

*era*

*ERASE* variable()

Effacer des tableaux et délibérer la place mémoire correspondante (chap. 6)

*ev* !!!

*EVERY* temps *GOSUB* procédure version 3.0

Appeler une procédure à intervalle régulier (chap. 15)

*EXIST* (nom du fichier)

Teste si un nom de fichier existe sur le disque ou la disquette (chap. 14)

*ex*

*EXIT IF*

Permet de quitter (proprement) une boucle (chap. 6)

*FALSE*

Valeur booléenne : logiquement faux = 0 (chap. 6)

*file*

*FILES* [masque] [TO fichier|imprimante]

Afficher l'index (complet) du disque ou de la disquette (chap. 15)

*filese*

*FILESELECT* masque, indicatif, nom du fichier

Ouvrir la boîte de choix de fichier (pour la sauvegarde ou le chargement) (chap. 14)

*fi*

*FILL* x,y

Remplir une surface avec différents motifs (chap. 9)

*FIX*(nombre)

Partie entière d'un nombre décimal (chap. 6)

@

*FN* Nom[(Argument)] version 3.0

Fonction définie par vous-même (chap. 7)

*f*  
**FOR** *compteur* = début **TO** | **DOWNTO** fin [**STEP** importance du pas]  
*instruction(s)*  
**NEXT** *compteur*  
 Boucle de comptage (chap. 5)

**FRAC**(*nombre*)  
 Partie après la virgule d'un nombre fractionnaire (chap. 6)

*fu !!! version 3.0*  
**FUNCTION** *nom*[(*paramètre*)]  
*[définition de la variable]*  
*instruction(s)*  
**RETURN** *valeur*  
**ENDFUNC**  
 Définition d'une fonction sur plusieurs lignes (chap. 8)

**GET** *X1,Y1,X2,Y2,string,variable*  
 stocke (un morceau de) l'écran sous forme de string (chap. 9)

**GET** *#n,(enregistrement)*  
 Permet de lire un enregistrement quelconque dans un fichier relatif.

*[g|@]*  
**GOSUB** *nom de la procédure*  
 Appeler une procédure dont le nom est "nom de la procedure" (chap. 8)

*got*  
**GOTO** *marque*  
 Sauter à un segment du programme (chap. 3)  
 Permet un branchement incondtionnel.

*i*  
**IF** *condition* [**THEN**]  
*instruction alors*  
**[ELSE**  
*instruction sinon]*  
**[ELSE IF ...]**  
**ENDIF**  
 (bloc d') *Instruction de choix* (chap. 2)  
 Branche le programme sur différents blocs programmes suivant leur valeur logique "vrai ou faux".

*INC variable*

Augmenter (incrémenter) une variable numérique d'une unité (chap. 5)

*inf*

*INFOW[#]numéro de fenêtre, texte*

Texte à faire figurer sur la ligne d'information (fenêtre GEM) (chap. 11)

*INKEY\$*

Identifier un caractère entré à l'aide du clavier (chap. 8)

*inp*

*INPUT ["remarque"][,], variable(s)*

Entrer des données depuis le clavier pendant l'exécution du programme (chap. 2)

*inp*

*INPUT #numéro de fichier, variable(s)*

Entrer des données du disque ou de la disquette vers le programme (chap. 14)

*ins !!! version 3.0*

*INSERT nom du tableau(index), élément*

Insérer un élément dans le tableau (chap. 15)

*INSTR (début, string source, string recherché)*

Position du string recherché dans un string source (chap. 8)

*INT(nombre)*

Fournit le nombre entier le plus approchant (inférieur ou égal) du nombre en question (chap. 6)

*ki*

*KILL nom du fichier*

Effacer un fichier sur le disque ou la disquette (chap. 15)

*LEFT\$ (string source, nombre)*

Partie gauche d'un string (chap. 8)

*LEN(string)*

Longueur d'un string (chap. 8)

*le*

*[LET] variable = valeur*

Affecter une valeur à une variable (chap. 6)

*li*

*LINE X1,Y1,X2,Y2*

relie les points (x1,y1) et (x2,y2) par une droite (chap. 9)

*li [li input]*

*LINE INPUT ["remarque"][:|,]string*

Lire un string depuis le clavier (chap. 13)

*li [li input]*

*LINE INPUT #numéro de fichier,string*

Charge un string depuis le disque ou la disquette (chap. 14)

*ll*

*LLIST*

Sortir le listing du programme sur l'imprimante (chap. 1)

*loa*

*LOAD nom du fichier*

Charger un programme contenu sur un fichier dans la mémoire de travail (chap. 1)

*loc*

*LOCAL variable(s)*

Déclarer une variable comme d'usage local (dans une fonction ou une procédure) (chap. 7)

*locat !!! version 3.0*

*LOCATE ligne, colonne*

Positionner le curseur sur la colonne et ligne spécifiées (chap. 7). Pour la version 2.0, on remplacera locate par print AT (colonne, ligne) qui permet d'afficher des données à un endroit précis de l'écran.

*l*

*LOOP*

voir DO

*lpr*

*LPRINT [expression][:;][:,]*

Sortir des données sur l'imprimante(chap. 13)

*MENU tableau()*

Reprendre un texte pour un menu déroulant (GEM) (chap. 11)

*me off*

**MENU OFF**

Remplacer dans son état normal une option pointée dans un menu (chap. 11)

**MENU(0)**

Option retenue dans un menu (GEM) (chap. 11)

**MID\$(string source,début [,nombre])**

Segment situé dans le string (chap. 8)

**MID\$(string cible,début [,nombre]) = string partiel**

Remplacer un string par des signes (chap. 8)

**Nombre MOD Nombre**

Reste d'une division d'un nombre entier (chap. 6)

*mou*

**MOUSE x,y,touche**

Demander la position du curseur ainsi que le statut des touches de la souris (chap. 9)

**MOUSEK**

Sert à tester la position actuelle de la souris ainsi que l'état des 2 boutons de la souris (chap. 8)

**MUL variable,quantité**

Multiplier une variable numérique par la quantité indiquée (chap. 5)

*na*

**NAME ancien nom AS nouveau nom**

Renommer un fichier sur le disque ou la disquette (chap. 15)

**NEW**

Effacer le programme dans l'éditeur (chap. 1)

*n*

**NEXT**

voir FOR

**NOT valeur version 3.0**

Inversion de l'expression logique qui suit (chap. 6)

**OFF**

voir MENU

*ON variable GOSUB liste des procédures*

Appeler par des valeurs numériques (choix multiple) (chap. 10)

*ON MENU*

Contrôler l'appel dans les menus (chap. 11)

*ON MENU GOSUB nom de la procédure*

Appeler les procédures par le menu GEM (chap. 11)

*o a [O"A"]*

*OPEN "A",#numéro du fichier,nom du fichier*

Ouvrir un fichier pour y ajouter des données (chap. 14)

*o i [O "I"]*

*OPEN "I",#numéro du fichier,nom du fichier*

Ouvrir un fichier pour y lire les données (chap. 14)

*o o [O"O"]*

*OPEN "O",#numéro du fichier,nom du fichier*

Ouvrir un fichier pour y écrire des données (chap. 14)

*o w*

*OPENW numéro de fenêtre*

Ouvrir une fenêtre GEM (chap. 11)

*opt base*

*OPTION BASE nombre*

Définir la limite inférieure (0 ou 1) pour l'index de tableau (chap. 6)

*Valeur OR valeur*

Opération logique OU sur deux conditions (chap. 6)

*pa*

*PAUSE cinquantième de seconde*

Interrompt l'exécution du programme pendant un certain temps (chap. 14)

*pb*

*PBOX x gauche,y haut,x droite,y bas*

Dessiner et remplir des carrés (chap. 9)

*pc*

*PCIRCLE x,y,rayon*

Dessiner et remplir des cercles (chap. 9)



*pe**PELLIPSE* *x,y,x\_rayon,y\_rayon*

Dessiner et remplir des ellipses (chap. 9)

*pl**PLOT* *x,y*

Dessiner des points (chap. 9)

*polyf**POLYFILL* *coins,x(),y()*

Dessiner et remplir des polygones (chap. 9)

*pol**POLYLINE* *coins,x(),y()*

Dessiner des polygones (chap. 9)

*prb**PRBOX* *x\_gauche,y\_haut,x\_droite,y\_bas*

Dessiner et remplir des carrés à coins arrondis (chap. 9)

*p**PRINT* [*AT(colonne,ligne);*][*expression*][*;*],]

Afficher des données sur l'écran à la colonne et ligne spécifiées (chap. 2)

*p**PRINT* *#numéro du fichier,données* [*;*]

Inscrire des données sur le disque dur ou la disquette (chap. 14)

*pro**PROCEDURE* *nom[(paramètre)]**définition de variables**instruction(s)**RETURN*

Définition d'une procédure (sous-programme) (chap. 7)

*psa**PSAVE* *nom du fichier*

Sauvegarder le programme sur le disque ou la disquette avec la protection LIST (chap. 15)

*PUT* *X1,Y1,variable-string*

Afficher à l'écran le contenu d'un string (chap. 9)

*qs !!! version 3.0*

*QSORT tableau( + | - ) [ WITH critère ( ) ]*

Trier les éléments d'un tableau d'après leur taille (quicksort) (chap. 15)

*q*

*QUIT*

Permet de quitter l'interpréteur (chap. 1)

*RANDOM(nombre)*

Fournir un nombre aléatoire entier (chap. 5)

*ra !!!*

*RANDOMIZE version 3.0*

Lancer le générateur de nombres aléatoires (chap. 5)

*rb*

*RBOXx\_gauche,y\_haut,x\_droite,y\_bas*

Dessiner et remplir un carré avec des coins arrondis (chap. 9)

*rea*

*READ variable[,...]*

Lire les éléments DATA et les transformer en variables (chap. 6)

*r*

*REM suite de caractères*

Commentaire (') (!) (chap. 2)

*ren !!! version 3.0*

*RENAME ancien nom AS nouveau nom*

Renommer un fichier (chap. 15)

*rep*

*REPEAT*

*instruction(s)*

*UNTIL condition*

Boucle de répétition avec test de la condition à la fin (chap. 4)

*res*

*RESTORE[marque]*

Ramener le pointeur DATA au début ou sur une marque spécifiée (chap. 13)

*ret !!!*

*RETURN[valeur]*

Fin de la procédure ou retour de la valeur sous FUNCTION(3.0) (chap. 7)

*RIGHT\$(string source, nombre)*

Partie droite d'un string (chap. 10)

*ru*

*RUN [nom du fichier]*

(Re)Lancer le programme (chap. 1)

*sa*

*SAVE nom du fichier*

Sauvegarder un programme sur le disque ou la disquette (chap. 1)

*s !!! Version 3.0*

*SELECT variable*

*CASE valeur [TO valeur]*

*[instruction(s)]*

*[DEFAULT*

*instruction(s)]*

*ENDSELECT*

Choix répétitif (chap. 10)

Permet des branchements en fonction de la valeur de l'expression numérique.

*set*

*SETCOLOR registre des couleurs,rouge,vert,bleu*

Placer du rouge, du vert et du bleu dans un registre (chap. 9)

*setd !!! version 3.0*

*SETDRAW x,y,angle*

Positionner le curseur-tortue et le diriger (chap. 9)

*sett*

*SETTIME heure,date*

Fixer l'heure et la date (chap. 8)

*sg*

*SGET variable-string*

Archiver le contenu de l'écran sous forme de chaîne de caractères (chap. 9)

*so*

*SOUND canal,puissance,note,octave,durée*

Produire des sons (chap. 9)

*SPACE\$(nombre)*

Produit une chaîne composée de (x) caractères blancs (chap. 8)

*spr*

*SPRITE* string de forme [*x,y*]

Définir des objets graphiques (sprites) (chap. 9)

*spu*

*SPUT* variable-string

Copie rapide d'une chaîne de 32000 caractères dans la mémoire écran (chap. 9)

*ss* !!! version 3.0

*SSORT* tableau( + | - ) [ *WITH* critère ( ) ]

Trier les éléments d'un tableau d'après leur taille (shellsort) (chap. 15)

*STEP*

voir FOR

*STR\$(nombre)*

Convertit une valeur numérique (x) en chaîne de caractères (chap. 6)

*STRING\$(nombre, caractère)*

String comme chaîne composée du même caractère (chap. 8)

*su*

*SUB* variable, quantité

Diminuer une variable numérique de la quantité indiquée (chap. 5)

*sw*

*SWAP* variable\_1, variable\_2

Echanger deux valeurs de variable (chap. 15)

*t*

*TEXT* x, y, [ *largeur*, ] *string*

Afficher un texte graphique (au pixel près) à partir du point de coordonnées (x,y) (chap. 9)

!!!

*THEN*

voir IF

*tim*

*TIME\$*

Indiquer l'heure-système [et mettre à l'heure(3.0)] (chap. 8)

*tit*

*TITLEW [#]numéro de fenêtre, texte*

Texte pour la ligne de titre (fenêtre du GEM) (chap. 11)

*TO*

voir FOR (et aussi CASE)

*to !!!*

*TOPW [#]numéro de fenêtre*

Active la fenêtre NO (chap. 11)

*TRUE*

Valeur booléenne : logiquement vrai = -1 (chap. 6)

*TRUNC(nombre)*

Partie entière d'un nombre décimal (chap. 6)

*u !!!*

*UNTIL*

voir DO | REPEAT

*UPPER\$ variable-string*

Convertir les minuscules d'un string en majuscules (chap. 8)

*VAL(string)*

Valeur numérique d'un string (chap. 6)

*!!!*

*VAR paramètre*

Paramètre variable dans une procédure ou une fonction (chap. 7)

*we*

*WEND*

voir WHILE

*w*

*WHILE condition*

*instruction(s)*

*WEND*

Boucle de répétition avec test de la condition au début (chap. 4)

*w !!!*

*WHILE*

voir DO | LOOP

!!! version 3.0

WITH

permet d'indiquer un critère de tri sous forme de tableau comportant au moins 256 éléments.

voir QSORT|SSORT

Valeur XOR valeur

Opération logique SOIT...SOIT sur deux conditions (chap. 6)

## 16.2. Du GFA 2.0 au GFA 3.0

1. Les programmes "BAS écrits en GFA Basic version 2.0 doivent tout d'abord être sauvegardés sur une disquette ou un disque dur par Save,A et devenir ainsi des fichiers avec extension LST. Merge permet alors de les charger dans l'éditeur du GFA Basic version 3.0 pour les traiter. Après quoi vous pouvez les sauvegarder en les faisant devenir des fichiers portant l'extension soit GFA soit LST. ATTENTION : l'interpréteur GFA ne comprend pas les lignes de programme commençant par la double-flèche = => !
2. Dans la version 3.0, on peut appeler les procédures directement par leur nom, sans GOSUB ni arobas (@). La définition de fonctions sur plusieurs lignes est rendue possible par FUNCTION (on les appelle par "@"). Les variables peuvent être transmises comme paramètres et être restituées modifiées après traitement par la procédure (ou la fonction).
3. La touche <Help> (ou <control> <Help>) permet le *repliage* (folding) des procédures dans la version GFA Basic 3.0, ce qui est très confortable lors de l'écriture de programmes longs comportant de multiples procédures.
4. La nouvelle version GFA Basic 3.0 offre une véritable arithmétique des nombres entiers, ce qui rend les opérations sur les nombres entiers beaucoup plus rapides qu'avec le GFA Basic version 2.0. On peut disposer de nombres longs (INTEGER - 4 byte - %), moyens (WORD - 2 byte - &) ou courts (BYTE - 1 byte - |).

Les nombres décimaux vont jusqu'à 8 bytes (seulement 6 dans le GFA Basic version 2.0), ce qui porte la précision après la virgule de 11 à 13 places et augmente le champ des valeurs de  $10^{154}$  à  $10^{308}$ .

## Chapitre 17

### Une mine de renseignements bien utiles

Il n'est pas rare de rencontrer des problèmes en programmant, mais ils peuvent vous faire perdre beaucoup de temps. Ils peuvent avoir des origines très diverses. Je ne peux pas vous aider s'il s'agit de découragement, d'aversion. Dans ces cas-là, il vaut mieux abandonner votre Atari pendant quelques temps, afin qu'il vous laisse en paix. Il m'est possible de vous aider si vous avez perdu le fil directeur de votre programme : vous manque-t-il un mot ? regardez ci-dessous, vous allez peut-être le trouver....

#### 17.1. Les mots clés du GFA Basic

##### *Affichage*

PRINT, LPRINT, AT, LOCATE(3.0)

##### *Boucles*

voir répétition

##### *Commentaire*

REM, ', !

### Conditions

IF, THEN, ELSE, ELSE IF(3.0), SELECT(3.0), CASE(3.0), DEFAULT(3.0), ENDIF, ENDSELECT(3.0), ON GOSUB, EXIT IF, AND, NOT, OR, XOR, =, <, <=, >, >=, <>, ==

### Définitions

DEFFN, FUNCTION(3.0), PROCEDURE, LOCAL, RETURN

### Données

DIM, DIM ?, OPTION BASE, ERASE, CLEAR, READ, DATA, RESTORE, LOCAL, !, | (3.0), &(3.0), %, #, \$, DEFBIT(3.0), DEFBYT(3.0), DEFINT(3.0) DEFFLT(3.0), DEFSTR(3.0), DEFWRD(3.0), DATE\$, TIME\$, SETTIME, SWAP, QSORT(3.0), SSORT(3.0), WITH(3.0)

### Ecran

CLS, LOCATE(3.0), PRINT, PRINT AT

### Entrée de données

INKEY\$, INPUT, LINE INPUT

### Fenêtre

voir GEM

### Fichier/disque

OPEN, CLOSE, EOF, INPUT #, PRINT #, DIR, FILES, FILESELECT, KILL, NAME, RENAME(3.0), LOAD, SAVE, PSAVE

### Fonctions

DEFFN, FN, FUNCTION(3.0), ENDFUNC(3.0), LOCAL, VAR(3.0), @,

voir entre autres mathématiques et strings



*GEM*

ALERT, FILESELECT, OPENW, CLOSEW, TITLEW, INFOW, TOPW(3.0),  
MENU, ON MENU, ON MENU GOSUB, MOUSE, MOUSEK, DEFMOUSE

*Graphique*

CLS, COLOR, SETCOLOR, DEFFILL, DEFLINE, DEFTXT, DEFMOUSE,  
SPRITE, PLOT, LINE, DRAW, FILL, BOX, RBOX, CIRCLE, ELLIPSE, POLYLINE,  
PBOX, PRBOX, PCIRCLE, PELLIPSE, POLYFILL, GET, PUT, SGET, SPUT

*Imprimante*

LPRINT, LLIST

*Mathématiques*

=, +, -, \*, /, \, DIV, MOD, ^, LET, INC, DEC, ADD, SUB, MUL, DIV, INT, FIX,  
ASC, CHR\$, VAL, STR\$, RANDOM, SWAP

*Musique*

SOUND

*Nombres*

voir données et mathématiques

*Procédure*

PROCEDURE, RETURN, LOCAL, GOSUB, @

*Tableaux*

voir données

*Programme*

REM, END, FN, GOSUB, GOTO, @, RUN, DEFFN, FUNCTION(3.0),  
ENDFUNC(3.0), PROCEDURE, RETURN, PAUSE

### Répétition

FOR, TO, DOWNT, NEXT, DO, LOOP, WHILE, UNTIL, WEND, REPEAT,  
EXIT IF, GOTO, <label>

### Sauts

GOTO, GOSUB, EXIT IF, <marque :>

### Sélection par menus

voir GEM

### Strings

CHR\$, STR\$, SPACE\$, STRING\$, VAL, LEN, INSTR, LEFT\$, RIGHT\$, MID\$, +,  
UPPER\$, DATE\$, TIME\$, INKEY\$

### Système

DIR, FILES, KILL, NAME, RENAME(3.0), ALERT, FILESELECT

## 17.2. Pour éviter les erreurs

1. Tous vos blocs sont-ils bien pourvus d'une marque de début et d'une marque de fin de bloc ? (sélection, boucle, procédure, fonction)
2. Les conditions introduites après IF, CASE, FOR, WHILE, UNTIL peuvent-elles être remplies ?
3. Avez-vous bien signalé le type de vos variables à l'aide de

!, &, |, %, #, \$ ?

4. Les variables transmises ont-elles bien le type et la valeur qu'il faut ? Le nombre des paramètres transmis correspond-il à celui des paramètres définis ? Si nécessaire, certains paramètres sont-ils bien définis comme étant variables ?

5. Avez-vous bien fait attention à l'utilisation des *petites choses* comme :

", ; = ( ) ' ! | & % # \$

L'éditeur filtre de lui-même pas mal d'erreurs, il ne vous laisse pas aller plus loin et vous bloque sur la ligne erronée. Si vous voulez reporter à plus tard la recherche de la cause de cette erreur afin de pouvoir écrire tout de suite la ligne suivante, placez une apostrophe (') au début de la ligne : vous vous débarrassez ainsi momentanément de ce tracas, mais attention, ce n'est que partie remise...

### 17.3. Pour conclure

Vous voilà à la fin de votre initiation au GFA Basic. J'ai dû en exclure une foule de termes très utiles de la version 3.0. Vous saurez très certainement maintenant vous débrouiller seul en ce qui concerne la programmation en GFA Basic, mais vous pourrez aussi vous appuyer sur le manuel et les autres livres existants.

Vous aurez bien sûr besoin de recourir encore à ce livre, et il ne peut pas être nuisible de le reprendre du début à la fin de temps en temps.



**Partie 5**

**Annexes**



# Annexe A

## Les commandes éditeur

### 1. Déplacement du curseur

<b>Touche du clavier</b>	<b>Effet obtenu</b>
Clic avec la souris(gauche)	Position quelconque
Flèche montante	Une ligne vers le haut
Flèche descendante	Une ligne vers le bas
Flèche à gauche	Un espace vers la gauche
Flèche à droite	Un espace vers la droite
< Control > < flèche à gauche >	Début d'une ligne
< Control > < flèche à droite >	Fin d'une ligne
< Control > < flèche montante >	Remonter
< Control > < flèche descendante >	Descendre
< ClrHome >	Début de page
< Control > < ClrHome >	Début du texte
< Control > -Z	Fin du texte
< Control > -G	Saut jusqu'à un no de ligne (3.0)

Il y a un contrôle de la syntaxe avant chaque changement de ligne !

### 2. Insérer et effacer

<b>Touche du clavier</b>	<b>Effet obtenu</b>
< F8 >	Ecraser en réécrivant dessus
< Insert >	Insérer une nouvelle ligne
< Control > -N	Insérer une nouvelle ligne(3.0)
< Control > < Delete >	Effacer une ligne
< Control > -Y	Effacer une ligne

< Control > -U	Rappeler une ligne après gommage (3.0)
< Delete >	Effacer le caractère suivant le curseur
< Backspace >	Effacer le caractère précédant le curseur

### 3. Manipulation des blocs

<u>Touche du clavier</u>	<u>Effet obtenu</u>
< Shift > < F5 >	marquer le début d'un bloc
< F5 >	marquer la fin d'un bloc
< F4 > + H(Hide)	ôter les marques du bloc
< F4 > + C(Copy)	copier le bloc
< F4 > + M(Move)	déplacer le bloc
< F4 > + < Control > -D	effacer le bloc
< F4 > + S(Start)	placer le curseur au début du bloc
< F4 > + E(End)	placer le curseur à la fin du bloc
< F4 > + L(List)	imprimer le bloc sur imprimante
< F4 > + W(Write)	sauvegarder le bloc sur une disquette
< F2 > (= Merge)	charger un bloc depuis une disquette (seulement pour les fichiers ".LST")

### 4. Autres opérations

<u>Touche du clavier</u>	<u>Effet obtenu</u>
< Tab >	curseur déplacé de 8 positions vers la droite
< Control > < Tab >	curseur déplacé de 8 positions vers la gauche
< Undo >	annulation des modifications faites dans une ligne lorsque le curseur s'y trouve encore;
< Help >	"pliage et dépliage" dans le folding
< Control > < Help >	folding d'une procédure (version 3.0)
< Control > -F	folding de plusieurs procédures situées après le curseur (version 3.0)
< Control > -R	recherche d'un segment de texte
< Control > -E	remplacement d'un segment de texte (2.0)
< Esc >	remplacement d'un segment de texte (3.0)
< Control > +	mode direct; annulation par < control >
< Shift > +	< shift de gauche > < alternate >
< Alternate >	mode direct, donne des notes de musiques en version 3.0
< Shift > < F3 >	mode direct, donne une ligne de +
< esc >	interruption du programme
	quitter l'éditeur et le GFA Basic 3.0
	mode direct version 2.0 ; annulation par < ESC > < RETURN >



## Annexe B

### Les menus du GFA Basic

#### 1. Pour appeler le programme :

Double-clic avec la touche gauche de la souris sur GFABASIC.PRG

#### 2. Le menu principal :

<F1>	Load	Chargement de programmes GFA-/BAS
<Shift><F1>	Save	Sauvegarde de programmes GFA-/BAS
<F2>	Merge	Chargement (par ajout) de programmes LST
<Shift><F2>	Save,A	Sauvegarde de programmes LST
<F3>	Llist	Impression du listing de votre programme
<Shift><F3>	Quit	Sortie du système GFA Basic
<F4>	Block	Menu des opérations sur les blocs :
C	Copy	-copier le bloc
M	Move	-déplacer le bloc
W	Write	-sauvegarder le bloc sur une disquette
L	Llist	-imprimer le bloc sur imprimante
S	Start	-rechercher le début du bloc
E	End	-rechercher la fin du bloc
<Control><D>	^Delete	-effacer le bloc
H	Hide	Oter les marques du bloc
<Shift><F4>	New	Effacer le programme dans l'éditeur
<F5>	BlkEnd	Marquer la fin d'un bloc
<Shift><F5>	BlkSta	Marquer le début d'un bloc
<F6>	Find	Appeler la routine de recherche
<Shift><F6>	Replac(e)	Appeler la routine de remplacement
<F7>	PgDown	Faire monter la page
<Shift><F7>	PgUp	Faire descendre la page
<F8>	Insert/Overwr	Insérer ou réécrire
<Shift><F8>	Txt16/Text8	23/48 lignes (seulement en monochrome !)
<F9>	Flip	Dernier affichage de l'écran
<Shift><F9>	Direct	Appeler le mode direct

<F10>	Test	Contrôle de la structure du bloc
<Shift><F10>	Run	Lancer le programme

### 3. La version 3.0 comprend en plus

#### les possibilités suivantes :

Le symbole Atari à gauche fait disparaître le menu principal et ouvre un menu déroulant :

<b>DESK GFA BASIC</b>		<b>accessoires</b>
Save		Sauvegarde d'un programme sur une disquette ou un disque dur dans des fichiers ".GFA"
Load		Chargement d'un fichier ".GFA"
Deflist		Fixer le mode d'écriture de vos programmes :
	0	L'éditeur écrit en lettres majuscules les mots du vocabulaire du GFA Basic et en minuscules les noms de variables, procédures et fonctions
	1	Les mots du vocabulaire Basic GFA et les noms commencent par une majuscule suivie de minuscules, le reste est écrit en minuscules.
	2	Comme sous DEFLIST 0 : de plus, les variables sans marque reçoivent le signe "#"
	3	Comme sous DEFLIST 1 : de plus, les variables sans marque reçoivent le signe "#" (définition automatique : 0 pour la version 3.0 et 1 pour la version 2.0)
Nouveaux noms		Savoir si de nouvelles variables vont être définies (prédéfinies : non)
Editor		Retour sous l'éditeur
























































A l'extrémité droite du menu principal, vous trouvez le numéro de ligne actuel (à partir de 0). L'heure est indiquée au-dessus, vous pouvez cliquer dessus pour la modifier.

Lors des sauvegardes avec Save ou Save,A, il y a confection d'un fichier de même nom portant l'extension ".BAK" (=Backup). Toutes les versions 2 du GFA Basic n'offrent pas forcément ces Backup !

# Annexe C

## Tableau des caractères ASCII de l'Atari

Les tableaux suivants vous donne les codes ASCII de l'Atari ainsi que les nombres décimaux.

0 = 	1 = 	2 = 	3 = 	4 = 
5 = 	6 = 	7 = 	8 = 	9 = 
10 = 	11 = 	12 = 	13 = 	14 = 
15 = 	16 = 	17 = 	18 = 	19 = 
20 = 	21 = 	22 = 	23 = 	24 = 
25 = 	26 = 	27 = 	28 = 	29 = 
30 = 	31 = 	32 = 	33 = 	34 = 
35 = 	36 = 	37 = 	38 = 	39 = 
40 = 	41 = 	42 = 	43 = 	44 = 
45 = 	46 = 	47 = 	48 = 	49 = 
50 = 	51 = 	52 = 	53 = 	54 = 

55 = [7]	56 = [8]	57 = [9]	58 = [!]	59 = [;]
60 = [<]	61 = [=]	62 = [>]	63 = [?]	64 = [e]
65 = [A]	66 = [B]	67 = [C]	68 = [D]	69 = [E]
70 = [F]	71 = [G]	72 = [H]	73 = [I]	74 = [J]
75 = [K]	76 = [L]	77 = [M]	78 = [N]	79 = [O]
80 = [P]	81 = [Q]	82 = [R]	83 = [S]	84 = [T]
85 = [U]	86 = [V]	87 = [W]	88 = [X]	89 = [Y]
90 = [Z]	91 = [I]	92 = [\]	93 = [J]	94 = [^]
95 = [_]	96 = [^]	97 = [a]	98 = [b]	99 = [c]
100 = [d]	101 = [e]	102 = [f]	103 = [g]	104 = [h]
105 = [i]	106 = [j]	107 = [k]	108 = [l]	109 = [m]
110 = [n]	111 = [o]	112 = [p]	113 = [q]	114 = [r]

115 = [s]	116 = [t]	117 = [u]	118 = [v]	119 = [w]
120 = [x]	121 = [y]	122 = [z]	123 = [{]	124 = [!]
125 = [}]	126 = [~]	127 = [Δ]	128 = [Ç]	129 = [U]
130 = [é]	131 = [ø]	132 = [W]	133 = [h]	134 = [ä]
135 = [ç]	136 = [ê]	137 = [Ë]	138 = [è]	139 = [Y]
140 = [î]	141 = [l]	142 = [N]	143 = [Å]	144 = [É]
145 = [æ]	146 = [Æ]	147 = [ô]	148 = [ü]	149 = [ö]
150 = [Q]	151 = [ù]	152 = [Y]	153 = [Ü]	154 = [U]
155 = [ç]	156 = [£]	157 = [Y]	158 = [Þ]	159 = [f]
160 = [A]	161 = [I]	162 = [O]	163 = [U]	164 = [R]
165 = [N]	166 = [a]	167 = [Q]	168 = [L]	169 = [r]
170 = [v]	171 = [k]	172 = [k]	173 = [l]	174 = [c]

175 = >	176 = ã	177 = ö	178 = ø	179 = œ
180 = œ	181 = œ	182 = ã	183 = ã	184 = ö
185 = "	186 = '	187 = ¢	188 = ¢	189 = ©
190 = ©	191 = ¢	192 = ¢	193 = ¢	194 = ¢
195 = ¢	196 = ¢	197 = ¢	198 = ¢	199 = ¢
200 = ¢	201 = ¢	202 = ¢	203 = ¢	204 = ¢
205 = ¢	206 = ¢	207 = ¢	208 = ¢	209 = ¢
210 = ¢	211 = ¢	212 = ¢	213 = ¢	214 = ¢
215 = ¢	216 = ¢	217 = ¢	218 = ¢	219 = ¢
220 = ¢	221 = ¢	222 = ¢	223 = ¢	224 = ¢
225 = ¢	226 = ¢	227 = ¢	228 = ¢	229 = ¢
230 = ¢	231 = ¢	232 = ¢	233 = ¢	234 = ¢

235 = ¢	236 = ¢	237 = ¢	238 = ¢	239 = ¢
240 = ¢	241 = ¢	242 = ¢	243 = ¢	244 = ¢
245 = ¢	246 = ¢	247 = ¢	248 = ¢	249 = ¢
250 = ¢	251 = ¢	252 = ¢	253 = ¢	254 = ¢
255 = ¢				

## CARACTERES DE CONTROLE

0 NUL	1 ♂ SOH	2 ♂ STX	3 ♂ ETX
4 ♂ EOT	5 ♂ ENQ	6 ♂ ACK	7 ♂ BEL
8 ✓ BS	9 ♂ HT	10 ♀ LF	11 ♀ VT
12 ♀ FF	13 ♀ CR	14 ♀ SO	15 ♀ SI
16 ♂ DLE	17 ♀ DC1	18 ♀ DC2	19 ♀ DC3
20 ♀ DC4	21 ♀ NAK	22 ♀ SYN	23 ♀ ETB
24 ♂ CAN	25 ♀ EH	26 ♀ SUB	27 ♀ ESC
28 ♀ FS	29 ♀ GS	30 ♀ RS	31 ♀ US



## **Annexe D**

### **Le GFA Basic V3.5**

#### **1. Introduction**

Cette nouvelle mise à jour vous propose, comme vous allez le constater, un bon nombre de fonctions supplémentaires et aussi des possibilités inédites ( et tant attendues!) de l'éditeur.

#### **2. L'éditeur**

Celui-ci conserve bien sûr la facilité d'emploi qui le caractérise et se permet même d'augmenter sa rapidité.

Voyons ce qu'il nous apporte :

- Il devient possible de replier (et déplier) les fonctions comme vous le faites pour les procédures, c'est-à-dire en vous positionnant sur **FUNCTION** avec le curseur, puis en appuyant sur **Help**.
- La recherche s'effectue maintenant dans les procédures et fonctions repliées.

### 3. Les nouvelles fonctions

Toutes, sauf deux, sont consacrées au calcul matriciel et vectoriel. Mathématiciens et physiciens seront comblés ! En voici la liste et leur description sommaire. Pour plus de précisions, veuillez vous reporter au manuel.

#### 3.1. Les fonctions matricielles

*MAT BASE 0*

*MAT BASE 1*

Initialisation du mode matriciel avec début d'indication à 1 ou 0.

*MAT CLR a()*

Initialise à 0 tous les éléments d'une matrice.

*MAT SET a()=x*

Initialise à x tous les éléments d'une matrice.

*MAT ONE a()*

Crée une matrice unité.

*MAT READ a()*

Place dans une matrice les éléments contenus dans des DATA.

*MAT PRINT [#i,]a()[,g,n]*

Ecrit tout ou partie d'une matrice dans un fichier (écran, imprimante ou autre).

*MAT INPUT #i,a()*

Lit une matrice dans un fichier.

*MAT CPY a([i,j])=b([k,l])[,h,w]*

Copie tout ou partie d'une matrice dans une autre.

*MAT XCPY a([i,j])=b([k,l])[,h,w]*

Copie la transposition de tout ou partie d'une matrice dans une autre.

*MAT TRANS a()[=b())*

Transpose une matrice ou copie sa transposée dans une autre.



*MAT ADD a()=b()+c()*

*MAT ADD a(),b()*

*MAT ADD a(),x*

Fonctions d'addition de matrices.

*MAT SUB a()=b()-c()*

*MAT SUB a(),b()*

*MAT SUB a(),x*

Fonctions de soustraction de matrices.

*MAT MUL a()=b()\*c()*

*MAT MUL x=a()\*b()*

*MAT MUL x=a()\*b()\*c()*

Fonctions de multiplication de matrices.

*MAT NORM a(),0*

*MAT NORM a(),1*

Fonctions de normalisation de matrices par lignes et par colonnes.

*MAT DET x=a([i,j])[n]*

Calcule le déterminant d'une matrice.

*MAT QDET x=a([i,j])[n]*

Calcule d'une manière optimisée le déterminant d'une matrice.

*MAT RANG x=a([i,j])[n]*

Calcule le rang d'une matrice.

*MAT INV a()=b()*

Place l'inverse d'une matrice dans une autre.

### 3.2. Les autres fonctions

*DATA*

Donne le contenu du pointeur des DATA. Renvoie 0 si le prochain READ doit renvoyer un 'out of DATA'.

*DATA=*

Positionne le pointeur des DATA à une valeur précédemment sauvegardée avec \_DATA.



**Annexe E****Index des mots-clés****A**

ADD .....	5-60
ALERT .....	10-132
ASC .....	6-79

**B**

BOX .....	9-112
-----------	-------

**C**

CASE .....	10-130
CHR\$ .....	6-79
CLOSE .....	14-179
CLOSEW .....	11-143
CLOSEW O .....	11-144
CLS .....	9-113
COLOR .....	9-115
Copie .....	1-6

# D

DATA .....	11-142
DATES\$ .....	8-100
DEC .....	5-58
Default .....	10-135
DEFFILL .....	9-114
DEFLINE .....	9-115
DEFMOUSE .....	9-118
DEFTEXT .....	9-117
DELETE .....	15-200
DIM .....	6-76
DIR .....	15-200
Direct .....	1-11
DIV .....	5-60
DO LOOP .....	4-45
DRAW .....	9-111, 9-119

# E

ELSEIF .....	2-21
END .....	2-20
Endselect .....	10-130
EOF .....	14-184
EXIT .....	14-180

# F

FILES .....	15-200
Fileselect .....	14-185
FILL .....	9-114
FIX .....	6-73
FOR .....	5-55
FOR TO NEXT .....	5-56
Formatage .....	1-5
Formater .....	1-4
FRAC .....	6-73

**G**

GET .....	9-116
GOTO .....	4-40

**I**

IF..ELSE..ENDIF .....	2-21
INC .....	5-58
INFOW .....	11-148
INKEY\$ .....	8-98 - 8-99
INPUT .....	2-15
Insert .....	1-9
INSERT .....	15-199
INSTR .....	8-103
INT .....	6-73

**K**

KILL .....	15-200
------------	--------

**L**

LET .....	6-70
Llist .....	1-11
Load .....	1-10
LPRINT .....	13-175

**M**

MENU OFF .....	11-150
MENU(0) .....	11-150
Merge .....	1-10
MID\$ .....	8-103
MOUSE .....	9-116
MOUSEK .....	8-99
MUL .....	5-60

## N

New .....	1-11
NEXT .....	5-55

## O

OPEN .....	14-178
OPEN o .....	11-143
OPENW .....	11-143

## P

PAUSE .....	14-185
PCIRCLE .....	9-113
PELLIPSE .....	9-113
POLYFILL .....	9-114
POLYLINE .....	9-113
PRINT .....	2-14, 2-23
PROCEDURE .....	7-84
PSAVE .....	15-203

## Q

QSORT .....	15-198
Quit .....	1-11

## R

RANDOM .....	5-54
RANDOMIZE .....	5-53 - 5-54
READ .....	11-142
REM .....	2-20
RENAME .....	15-200
REPEAT..UNTIL .....	3-31
RESTORE .....	13-164
ROUND .....	6-73
Run .....	1-11

**S**

Sauvegarde .....	1-4
Save .....	1-10
Save,A .....	1-10
SELECT .....	10-130
SETCOLOR .....	9-115
SETDRAW .....	9-120
SETTIME .....	8-100
SGET .....	9-116
SORTIR .....	1-6
SOUND .....	9-120
SPACE\$ .....	8-100, 13-166
SPUT .....	9-116
SSORT .....	15-198
STRING\$ .....	8-100
SUB .....	5-60
SWAP .....	15-200

**T**

Test .....	1-11
TIMES\$ .....	8-100
TITLEW .....	11-148
TRUNC .....	6-73

**U**

UPPER\$ .....	8-100
---------------	-------

**V**

VAR .....	15-193
Vide .....	10-131

# W

WAVE .....	9-121
WHILE WEND .....	4-45
WITH .....	15-199



ATARI ST+STE

## BIEN DEBUTER EN GFA BASIC

SCHUMANN

Connaissez-vous le GFA Basic, le langage le plus évolué sur ATARI ST+STE? N'avez-vous jamais rêvé de concevoir des applications performantes, des programmes élaborés, tirant le meilleur de votre machine? Alors voici le livre qu'il vous faut. A l'aide de nombreux exemples clairs et progressifs, BIEN DEBUTER EN GFA BASIC explique le vocabulaire fondamental de ce langage. Effectuez rapidement vos premiers pas en programmation, découvrez les boucles, les procédures, la manipulation des données, la création graphique et sonore... A chacune de vos idées, vous trouverez rapidement l'outil approprié, à chacun de vos problèmes, une aide précieuse et détaillée grâce à un glossaire complet de toutes les fonctions du langage. Progressivement vous apprendrez à utiliser l'éditeur, les notions de base du GFA Basic, puis l'ensemble des commandes du langage. Enfin profitez des nombreuses astuces et sachez éviter les pièges lors de vos développements.

### Principaux points traités :

- Introduction à l'éditeur du GFA Basic.
- Instructions et déclaration de variables.
- Saisie et sortie de données.
- Programmation de boucles à l'aide d'exemples.
- Découverte de la programmation structurée.
- Premières applications graphiques.
- Manipulation de différents fichiers.
- Réalisation d'un petit programme de base de données.
- Comment éviter les pièges et les erreurs communes.
- Convertir des programmes du GFA Basic 2.0 vers la version 3.0.
- Glossaire complet de toutes les fonctions du langage.



Réf. : ML 527. Prix : 129 F.  
ISBN 2-86899-169-6 / ISSN 0980-1928.

**EDITIONS MICRO APPLICATION**

58, RUE DU FAUBOURG POISSONNIERE  
75010 PARIS. TÉL.: (1) 47 70 32 44